www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

Agora com + páginas e com duas grandes novidades!

- Artigos de PHP
- e muito mais artigos para iniciantes em Delphi

DevMedia GROUP

Ano 7 · Edição 93 · R$9,90

ISSN 1517990-7

Desenvolimento Web & POO

Técnicas de orientação a objetos para incrementar suas aplicações com Delphi for PHP

NA WEB

Confira no portal ClubeDelphi duas vídeo-aulas sobre controle de versão de código-fonte!

Avançado
Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 1

ASP.NET
Veja como criar um sistema on-line de controle para uma vídeo-locadora – Parte 1

Coluna Ask the Expert

Desenvolvendo uma Aplicação Completa com PocketStudio
Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 2

Generics no RAD Studio 2007
Entendendo e utilizando Generics em suas aplicações .NET

Coluna: QuickUpdate

Easy Delphi
Saiba tudo sobre o componente StatusBar e personalize-o como desejar

PHP
Aprenda como melhorar a aparência de seu site com CSS no Delphi for PHP

Easy Delphi
Trabalhando com Menus, ActionList e ToolBars

©2007 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Studio, o logo do Visual Studio e "Seu potencial. Nossa inspiração" são marcas comerciais, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou em outros países. Os nomes das companhias ou produtos reais aqui mencionados podem ser marcas comerciais de seus respectivos proprietários.

Seu potencial. Nossa inspiração.
Microsoft®

Sumário

Olá, eu sou o DevMan! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

QuickUpdate

10 – QuickUpdate
Suporte técnico a distância
[*Adriano Santos*]

Expert Delphi Win32

12 – Streams
Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 1
[*Gustavo Chaurais*]

Tutorial Mini-Curso

20 – Desenvolvendo uma Aplicação Completa com PocketStudio
Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 2
[*Ricardo C. Boaro*]

Vanguarda Expert Delphi .NET

28 – Generics no RAD Studio 2007
Entendendo e utilizando Generics em suas aplicações .NET
[*Manoel Edesio B da Silva*]

Web Mini-Curso Delphi .NET

34 – Controle on-line de vídeo-locadora - Parte 1
Veja como criar um sistema on-line de controle para uma vídeo-locadora
[*Maikel Marcelo Scheid*]

Easy Delphi Win32

44 – Conhecendo e personalizando o componente StatusBar
Saiba tudo sobre o componente StatusBar e personalize-o como desejar
[*Adriano Santos*]

Easy Delphi Win32

52 – Menus, Ações e Atalhos
Trabalhando com Menus, ActionList e ToolBars
[*Maikel Marcelo Scheid e Edinei Daniel Steffen*]

Delphi PHP

56 – Orientação a Objetos no Delphi for PHP
Como aplicar conceitos de POO em aplicações PHP?
[*Rodrigo Carreiro Mourão*]

Delphi PHP

62 – Usando páginas de estilo (CSS)
Aprenda como melhorar a aparência de seu site com CSS no Delphi for PHP
[*Adriano Santos*]

Você percebeu os nomes ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

ickUpdate] Esse tipo de artigo "atualiza" o leitor rapidamente e um determinado assunto, sem entrar muito a fundo

lphi PHP] Artigo com foco na versão PHP do Delphi

torial] Artigo no estilo tutorial passo a passo

lphi Win32] Artigo com foco nas versões Win32 do Delphi

[Vanguarda] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado

[Easy] Artigos para iniciantes em Delphi

[Mini-Curso] Artigo que faz parte de um mini-curso.

[Delphi .NET] Artigo com foco nas versões .NET do Delphi

ClubeDelphi

Ano 8 - 93ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Ronan Almeida
dalmak@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Apoio
ASSESPRO PARANÁ

EDITORIAL

O Delphi for PHP representou uma revolução no conceito de desenvolvimento Web com PHP. Aliado a um IDE RAD, temos uma linguagem orientada a objetos (a partir do PHP5) e uma VCL (a VCL Web) com centenas de componentes e classes que podem ser utilizadas para desenvolver aplicações para internet de forma produtiva, RAD e com qualidade. Acredito que muitos leitores já estejam familiarizados com os principais conceitos da Programação Orientada a Objetos (POO) com o Delphi, como herança, polimorfismo, encapsulamento, mas como aplicar essas técnicas no Delphi for PHP, onde temos uma outra linguagem de "fundo"? Nesta edição, damos especial atenção ao desenvolvimento Web com Delphi for PHP focando na orientação a objetos. Você aprenderá como aplicar importantes conceitos na prática, permitindo que suas aplicações Web sejam mais robustas, mais modulares, com manutenção facilitada, com melhor impacto na mudança de um requisito, e tudo aquilo mais que a OO tem a oferecer. Vale lembrar que esta é uma série de artigos.

Este mês iniciamos uma nova coluna destinada aos leitores avançados e já começamos em grande estilo. Gustavo Chaurais desmestifica esse assunto que causa "arrepios" em muitos: Streams. Em seu artigo, dividido em duas partes, veja como implementar compactação, download em múltiplos pacotes e resources em suas aplicações.

Quando utilizamos uma coleção tipo TList no Delphi, podemos colocar dentro dela qualquer tipo de objeto. Porém, há um overhead ao se fazer esse procedimento, porque normalmente essas coleções aceitam tipos TObject ou Pointer (qualquer coisa) e se torna necessário fazer um typecast para o tipo que desejamos realmente manipular. Com o RAD Studio 2007, temos agora o recurso de Generics, que permite parametrizar em tempo de compilação qual o tipo de dado será colocado na coleção. Isso mesmo! Confira no artigo do Manoel.

E temos muito mais nesta edição: muito Easy Delphi para os iniciantes e Delphi for PHP. Temos também a continuação do mini-curso de Ricardo Boaro que trata do desenvolvimento para o PalmOS com PocketStudio. Temos duas grandes novidades. Uma nova coluna chamada "QuickUpdate" vai ajudar você leitor a se atualizar rapidamente, sobre assuntos importantes no dia-a-dia de um developer, através de uma linguagem simples, clara e direta. E no interior de cada artigo, você vai encontrar as "Notas do DevMan", que são dicas preciosas dadas pelo nosso mascote, que ajudarão você a entender ainda mais o assunto proposto por um determinado autor.

Sucesso com o Delphi.

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 550 Vídeo Aulas!
- 7 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: HD883**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

ssinatura
ClubeDelphi PLUS
ais conteúdo .NET por menos!

Informativo ClubeDelphi

ortal ClubeDelphi
ww.clubedelphi.net/portal

+550 vídeo aulas e 7 cursos online

Caro Leitor

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 550 vídeo aulas; ii) 7 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Brinde na web desta edição

2 Vídeos

Confira no portal ClubeDelphi PLUS 2 vídeo aulas sobre Controle de Versão com JEDI VCS

Controle de Versão com JEDI VCS:
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7662
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7663

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Últimas Vídeo-Aulas

Aprenda a desenvolver sistemas para o sistema operacional PalmOS
Acompanhe as aulas de Ricardo Boaro que falam unicamente do desenvolvimento de aplicações para PalmOS utilizando a IDE PocketStudio que é bastante semelhate ao Delphi inclusive utilizando-se de linguagem Pascal.

Curso Aplicação ASP.NET com Delphi e SQL Server 2005 Express-Parte III-Diferenças de sintaxes para Stored Procedures e Triggers
Veja nessa vídeo aula de Luciano Pimenta as principais diferenças de sintaxe para Stored Procedures e Triggers entre FireBird e SQL Server 2005.

Desenvolvendo uma aplicação para PalmOS com PocketStudio - Partes XI a XII
Veja nessas vídeo-aulas de Ricardo Boaro, como trabalhar com aplicações PalmOS com o PocketStudio.

Mini-Curso Controle de Versão com JEDI VCS
Veja nesse mini-curso de Adriano Santos como trabalhar com esta fabulosa ferramenta para controle de versão e gerenciamento de equipes.

Crie uma loja virtual com Delphi for PHP-Continuando com a página do carrinho de compra- Parte IX
Acompanhe a continuação do curso de Delphi for PHP do Fabrício Desbessel.

Iniciando no Delphi-Trabalhando com dbExpress-Parte I e II
Veja nessas vídeos de Guinther Pauli como trabalhar com dbExpress desde o início, conceitos e detalhes dos componentes de acesso.

Iniciando no Delphi – Criando componentes
Veja nessa vídeo aula de Guinther Pauli para iniciantes, como criar componentes para uso em suas aplicações.

NOVOS PLANOS

Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional** 1

Windows Server 2003

1 Registro de domínio grátis*
3 Domínios por conta
50 GB de transferência
30 GB para e-mails com SSL
1 GB de espaço
ASP, ASP.NET 2.0 e PHP 5
Access e MySQL 5
SQL Server 2005 Express
Painel de controle e webmail
Editor de páginas on-line
Anti-spam e Anti-vírus

R$ 25,90 por mês

30 DIAS GRÁTIS
PARA EXPERIMENTAR
Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

RVSTNET

DOMÍNIO GRÁTIS
ENQUANTO FOR CLIENTE
Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

PAINEL DE CONTROLE
PLESK 8.1 PROFISSIONAL
Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

www.hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Usando o Scroll do mouse para "rolar" o DBGrid

Olá, há tempos que venho procurando uma forma de fazer com que o *scroll* do mouse funcione nos *DBGrid's* do meu sistema, é possível?

Marcelo Freitas

Olá Marcelo, respondendo a sua questão: sim, claro qu existe como criar esse recurso. O que precisamos fazer é implementar uma nova classe no formulário, ou em uma *Unit* de funções e fazer uso dela. Crie uma nova classe no seu formulário como segue:

```
  TWheelDBGrid = class(TDBGrid)
  public
    property OnMouseWheel;
  end;
```

Em seguida, declare o evento *DBGridMouseWheel* na seção public do formulário, como podemos ver logo em seguida:

```
…
public
  { Public declarations }
  procedure DBGridMouseWheel(Sender: TObject;
    Shift: TShiftState; WheelDelta: Integer;
    MousePos: TPoint; var Handled: Boolean);
end;
```

Logo abaixo da seção *Implementation* inclua a função a seguir:

```
function GetNumScrollLines: Integer;
begin
  SystemParametersInfo(SPI_
GETWHEELSCROLLLINES,
    0, @Result, 0);
end;
```

E por fim implemente o código do evento que criamos conforme a **Listagem 1**. Já no evento *OnCreate* do formulário inclua a linha:

```
TWheelDBGrid(DBGrid1).OnMouseWheel :=
  DBGridMouseWheel;
```

Panel

Por gentileza, gostaria de melhor o visual de minhas aplicações. Uma coisa legal seria criar um *TPanel* com cantos arredondados, tem como?

Flávio

Olá Flávio, há como, mas temos criar isso usando um *TImage* como auxiliar. Veja como fazer na **Listagem 2**. Na **Figura 1** podemos ver o resultado.

Figura 1. Panel com cantos arredondados

Listagem 1. Código do evento de scroll

```
procedure TForm1.DBGridMouseWheel(Sender: TObject;
  Shift: TShiftState; WheelDelta: Integer; MousePos:
  TPoint; var Handled: Boolean);
var
  Direction: Shortint;
begin
  Direction := 1;
  if WheelDelta = 0 then
    Exit
  else if WheelDelta > 0 then
    Direction := -1;
  with TDBGrid(Sender) do
  begin
    if Assigned(DataSource) and Assigned(DataSource.DataSet) then
      DataSource.DataSet.MoveBy(Direction * GetNumScrollLines);
    Invalidate;
  end;
end;
```

Listagem 2. Código para criar TPanel com cantos arredondados

```
procedure TForm1.FormCreate(Sender: TObject);
const
  bgcolor = $0080FF80;
  linecolor = clBlack;
var
  img: array of TImage;
  reg: hrgn;
  i: Integer;
begin
  for i := 0 to ComponentCount - 1 do
  begin
    if Components[i].ClassName = 'TPanel' then
    begin
      setlength(img, Length(img) + 1);
      img[i] := TImage.Create(Self);
      img[i].Width := (Components[i] as TPanel).Width;
      img[i].Height := (Components[i] as TPanel).Height;
      img[i].Parent := (Components[i] as TPanel);
      img[i].Canvas.Brush.Color := bgcolor;
      img[i].Canvas.pen.Color := bgcolor;
      img[i].Canvas.Rectangle(0,0,img[i].Width, img[i].Height);
      img[i].Canvas.pen.Color := linecolor;
      img[i].Canvas.RoundRect(0,0,img[i].Width - 1,img[i].Height - 1,20,20);
      reg := CreateRoundRectRgn(0,0,(Components[i] as TPanel).Width,
        (Components[i] as TPanel).Height, 20,20);
      setwindowrgn((Components[i] as TPanel).Handle, reg, True);
      deleteobject(reg);
    end;
  end;
end;
```

POR QUE A ALOG
É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB ACCREDITED BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos que "atualizam" o leitor rapidamente sobre um determinado assunto.

Quick Update

Suporte técnico a distância

Olá, sejam bem-vindos a nova coluna da revista ClubeDelphi, a coluna QuickUpdate. E vou começar dando uma dica para melhorar o suporte técnico à distância em sua empresa.

Quando se tem clientes em outras cidades e/ou estados fica bastante difícil deslocar-se ao cliente com certa freqüência, até mesmo porque muitas vezes o suporte é básico e não requer visita técnica, por isso muitos desenvolvedores utilizam-se de ferramentas como VNC, Logme-in ou mesmo o esquema de conexão remota do Windows XP/2000/2003. Porém um dos maiores problemas acaba acontecendo quando o computador do cliente possui roteador na rede e nem sempre sabe como desativar uma determinada porta ou mesmo configurar o Windows para receber a conexão remota.

Recentemente minha equipe e eu descobrimos um esquema gratuito e bastante interessante. Utilizando o Ultra-VNC (*www.uvnc.com/index.html*), software gratuito para conexão remota, é possível criar nosso próprio sistema de suporte on-line. A maior vantagem é que o programa gerado pelo Ultra-VNC é quem faz a conexão com o micro do suporte técnico, e não o contrário, ou seja, seu cliente que requisita uma conexão remota. Desta forma não há necessidade de qualquer configuração por parte do cliente, já que normalmente não há bloqueios para conexão externa.

Será necessário apenas fazer a configuração do roteador em sua empresa de forma que aponte para o micro de cada suporte técnico disponível.

Exemplo: digamos que você possua quatro máquinas para suporte técnico com os IP's 192.168.0.100 até 192.168.0.104. Configure em seu roteador para que as portas 30010, 30011, 30012 e 30013 sejam direcionadas para estes IP's. Em seguida instale o Ultra-VNC em cada uma dessas máquinas e crie um atalho para o Ultra-VNC Viewer no Desktop. Nas propriedades do atalho, em Destino("target"), acrescente o texto *–listen 30010* e repita esse passo para cada máquina, mudando apenas a porta. Um exemplo completo ficaria como:

```
"C:\Program Files\UltraVNC\vncviewer.exe"
-listen 30010
```

Pronto, agora será necessário baixar o arquivo de exemplo do site do Ultra-VNC e preparar-se para criar o executável final. Acesse o link *www.uvnc.com/addons/singleclick.html*. Aqui estão todas as instruções para criação do executável de suporte, mas vamos lá. Baixe e descompacte o arquivo de personalização da ferramenta no link

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

www.uvnc.com/custom.zip.

Perceba que foram descompactados os seguites arquivos:

- background.bmp: Arquivo que forma a borda do executável;
- helpdesk.txt: Possui as configurações necessárias para conexão;
- icon1.ico: Ícone comum do aplicativos;
- icon2.ico: Ícone quando conectado;
- logo.bmp: Logotipo que irá aparecer na janela;
- rc4key.reg: Arquivo de registro do programa no regedit.exe (Opcional).

Você pode criar seu próprio background, ícone para exibição e para conexão e também o logotipo. A única premissa é manter os mesmos nomes de arquivo. Abra o arquivo helpdesk.txt e perceba que há uma série de configurações. Preencha-as corretamente e salve. O mais importante é configurar corretamente o número do IP e porta no arquivo. Elas devem coincidir com as alterações feitas no roteador. Veja um exemplo de arquivo preenchido.

```
[TITLE]
Devmedia Suporte ONLine

[HOST]
Adriano Santos
-connect 205.201.12.240:30010 -noregistry

[HOST]
Guinther Pauli
-connect 205.201.12.240:30011 -noregistry

[HOST]
Gladstone Matos
-connect 205.201.12.240:30012 -noregistry

[HOST]
Suporte 4
-connect 205.201.12.240:30013 -noregistry

[TEXTTOP]
Clique duas vezes para solicitar conexão

[TEXTMIDDLE]
Antes de fazer a conexão entre em contato

[TEXTBOTTOM]
Fale com nosso técnico

[TEXTRBOTTOM]
[TEXTRMIDDLE]
[TEXTRTOP]
[TEXTBUTTON]
Mais informações

[WEBPAGE]
http://www.devmedia.com.br

[TEXTCLOSEBUTTON]
Parar teste

[BALLOON1TITLE]
Estabelecendo conexão

[BALLOON1A]
Tentativa de período 5 mim

[BALLOON1B]
Se falhar tente novamente

[BALLOON1C]

[BALLOON2TITLE]
Conexão ativada

[BALLOON2A]
[BALLOON2B]
[BALLOON2C]

[WEBPAGE]
http://www.devmedia.com.br
```

Note que o IP indicado não é o IP local e sim o externo seguido da porta de conexão. Feito isso, basta compactar todos os arquivos com o nome que ficará seu executável final, exemplo "Suporte.zip", que mais tarde se tornara *Suporte.exe*.

Agora entre no link *www.uvnc.com/pchelpware/creator/index.html* e preencha os campos do primeiro formulário (**Figura 1**). Digite usuário("foo") e senha("foobar"). Selecione o arquivo *Suporte.zip* e faça o *upload*. O site retornará um arquivo executável para download com o nome *Suporte.exe*. Faça o download e teste-o. (**Figura 2**)

Agora basta distribuir o executável e instruir seus usuários a utilizar. Toda vez que um usuário clicar duas vezes no nome de um dos técnicos de suporte, o mesmo receberá uma pergunta se deseja ou não aceitar a requisição. Aceitando-a, o técnico passa a ter controle total da máquina do cliente.

Com isso sua empresa ganhará um bom aliado no dia-a-dia. Boa sorte e até a próxima. ●

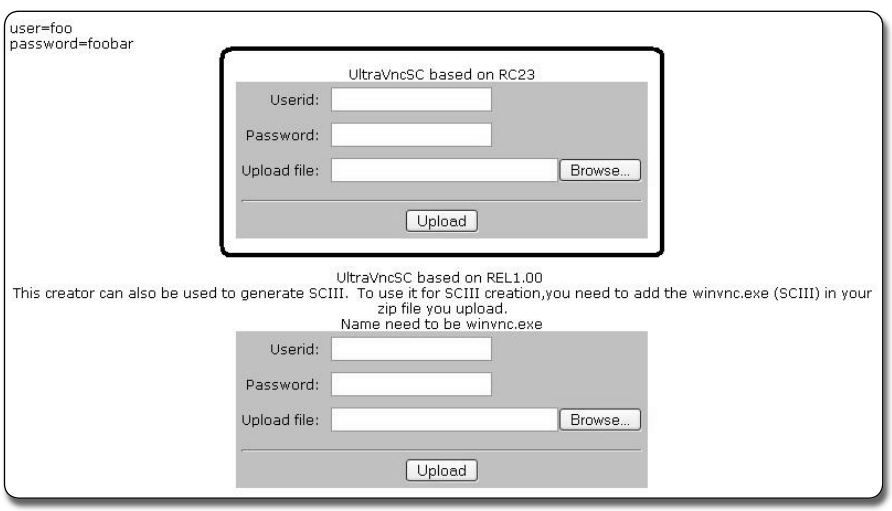
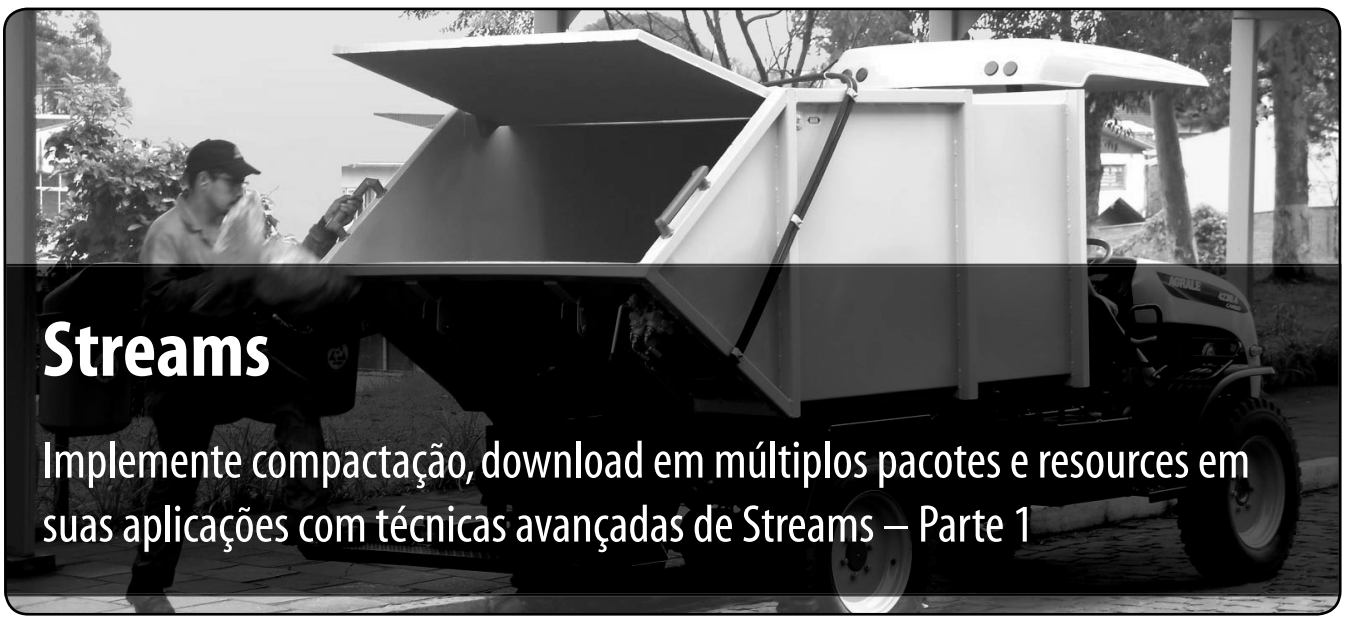


Figura 1. Fazendo upload

Figura 2. Programa de suporte

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET



Streams

Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 1

Gustavo Chaurais
(gustavoc@macrovision.com)
é Borland Delphi 7 Advanced Certified, Borland Delphi 2005 for Win32 Certified, Borland Delphi 2006 for Win32 Certified e Borland Delphi Instructor Certified. Foi palestrante das três últimas edições da Borland Conference Brasil e de outros grandes eventos nacionais. Além disso, é membro da coordenadoria do GIESSC. Hoje, ministra cursos, presta consultoria e atua como Software Engineer do projeto InstallAnywhere para a empresa norte-americana Macrovision Corporation.

Aí está o grande pesadelo de grande parte dos programadores Delphi. Muitos, quando ouvem falar em *streams,* desistem do código na hora! Mas, por que será que causa tanto medo? Talvez, isso se deva ao fato de estarmos trabalhando diretamente com *bits* e *bytes*. E isso nos remete aos "malucos escovadores de bits" de décadas atrás. Hoje, temos orientação a objetos, a adoção do *garbage collector* pelas mais diversas linguagens, dentre outros conceitos e técnicas que deixam nossa vida muito mais fácil. Por isso, desacostumamos com o fato de que, por trás de tudo, ainda temos os mesmos ponteiros, *bits* e *bytes*.

Neste artigo, veremos que *streams* são, na verdade, muito simples e podem ser utilizados por qualquer programador. Todos os que já trabalharam com *arrays*, listas (*TStringList*, *TObjectList*), *datasets*, ou qualquer tipo de coleção, não devem encontrar dificuldade para utilizar todos os recursos da classe *TStream*.

Com exemplos práticos e objetivos, abordaremos também dois assuntos extremamente relacionados: *Compression* e *Resources*. Veremos que não é necessário fazer o download nenhuma biblioteca para trabalhar com arquivos compactados, uma vez que o Delphi já implementa a sua. Construiremos, passo a passo, uma aplicação para compactação de múltiplos arquivos.

Quanto aos *Resources*, além do exemplo clássico de embutir uma DLL dentro de seu executável, aprenderemos também, a fundo, a alterar os recursos de outras aplicações, bem como a modificar suas *resourcestrings*. Enfim, tudo o que há de mais avançado no assunto.

Streams

Imagine um *dataset* bidirecional, como o *TClientDataSet* ou o *TTable*. Porém, em vez de armazenar registros e campos de um banco de dados, armazenaremos *bytes*, um após o outro, de um arquivo

que está no disco. Podemos pegar seu tamanho final, temos a possibilidade de navegar entre os *bytes*, etc., como se fossem registros.

Streams podem ser entendidos como uma seqüência de *bytes* na memória. Esses *bytes* podem ser tanto de um programa que foi carregado, como de uma *string* qualquer, um arquivo de recurso, ou como uma seqüência de *bytes* qualquer que desejamos representar. Analisemos código presente na **Listagem 1**.

A classe *TFileStream* herda de *TStream* e serve para representarmos arquivos do disco na memória. Seu uso é muito simples. No seu construtor, devemos passar: o caminho para o arquivo a ser representado e o modo de abertura desse arquivo. (ver nota "Modos de Abertura de Arquivos")

Vale lembrar que o arquivo ficará bloqueado para outros *streams* (e outros programas) até que o *stream* seja fechado. Como não existe um método *Close*, a única maneira de fazermos isso é chamando seu destrutor (ou *Free*). Portanto, é muito importante fechar o *stream* para que o arquivo seja liberado.

No exemplo citado, estamos criando dois *streams* do tipo *TFileStream* e chamando o método *CopyFrom* do que está criando o arquivo. Portanto, estamos simplesmente criando uma cópia do arquivo. Além disso, notamos que esse método necessita da quantidade de *bytes* que gostaríamos de copiar. No caso, utilizamos o arquivo inteiro (*Size*). Vejamos agora outro tipo de *stream*, também descendente de *TStream*, o *TStringStream*. Vamos modificar a entrada de dados para uma *string* na memória. (**Listagem 2**)

A classe *TStringStream* se difere da anterior apenas pela fonte de dados, que agora é uma *string*. Esta classe possui uma propriedade especial *DataString*, pela qual é possível sabermos qual a *string* está sendo armazenada na memória. O resultado deste exemplo será um arquivo texto contendo: *string* no *stream*.

Listagem 1. Trecho de código utilizando Streams

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileStreamIn: TFileStream;
  FileStreamOut: TFileStream;
begin
  FileStreamIn := TFileStream.Create('caminho para arquivo de entrada',
    fmOpenRead);
  FileStreamOut := TFileStream.Create('caminho para arquivo de saída',
    fmCreate);
  try
    FileStreamOut.CopyFrom(FileStreamIn, FileStreamIn.Size)
  finally
    FileStreamIn.Free;
    FileStreamOut.Free;
  end;
end;
```

Listagem 2. Trecho de código utilizando Streams

```
procedure TForm1.Button1Click(Sender: TObject);
var
  StringStreamIn: TStringStream;
  FileStreamOut: TFileStream;
begin
  StringStreamIn:= TStringStream.Create('string no stream');
  FileStreamOut:= TFileStream.Create('caminho para arquivo de saída',
    fmCreate);

  try
    FileStreamOut.CopyFrom(StringStreamIn, StringStreamIn.Size);
  finally
    StringStreamIn.Free;
    FileStreamOut.Free;
  end;
end;
```

Modos de Abertura de Arquivos

fmCreate
Caso o arquivo não exista, será criado. Caso exista, será zerado para que se comece um novo.

fmOpenRead
Apenas para leitura.

fmOpenWrite
Apenas para escrita.

fmOpenReadWrite
Leitura e escrita.

Cabe aqui apresentar as duas propriedades mais importantes de qualquer *stream*: *Position* e *Size*. Como em um *dataset*, *streams* possuem um cursor interno para a leitura e a escrita. No exemplo anterior, não precisamos lidar com tal cursor porque o método *CopyFrom* fez todo o trabalho. No entanto, se lermos o valor de *Position* antes de chamá-lo, veremos que seu valor é *0*. Após a cópia, este muda para o número de *bytes* copiado (no caso, o tamanho do arquivo). E a propriedade *Size*, como era de se esperar, retorna o tamanho do que foi carregado no *stream*. Ambas são do tipo *Int64*, um tipo especial de inteiro capaz de trabalhar com números de até *64bits*.

Experimente acrescentar a seguinte linha logo após a criação do *TStringList*:

```
StringStreamIn.Position := 7;
```

Ao rodar o programa, você verá uma exceção, pois como estamos tentando copiar todo o tamanho do arquivo (*StringStreamIn.Size*) e começando da posição *7*, ele tentará ler um conteúdo que não existe. Para corrigir isso, modifique a linha de cópia para:

```
FileStreamOut.CopyFrom(StringStreamIn,
  StringStreamIn.Size - StringStreamIn.
  Position);
```

Agora, a cópia será feita a partir da posição *7*, ou seja, o conteúdo final do arquivo será: *no stream* Além do método *CopyFrom* a maneira mais comum de lermos e escrevermos algo em um *stream* é utilizando seus métodos *Read* e *Write*. Analisemos o código da **Listagem 3**.

Existem diversas maneiras de manipular dados com *Read* e *Write*, porém, consideramos esta a mais simples. A primeira coisa a ser feita é definirmos um *buffer*. Um *buffer* precisa ser um local da memória livre para armazenarmos *bytes*. Neste caso, poderíamos utilizar um ponteiro para algum lugar (como um *PChar*) e trabalharmos com seu valor (variavel^), alocando e desalocando memória para ele (*GetMem*), ou um simples *array* (de *Byte*, *Char* ou outra estrutura de 8 bits), que também é uma região reservada da memória. Não poderíamos utilizar um *string* porque, apesar de também representar uma região alocada da me-

mória, possui algumas informações no primeiro caractere que não interessam neste caso. Portanto, decidimos utilizar um *array*.

O tamanho do *buffer* você é quem escolhe. Estamos trabalhando com 1024 *bytes* (0..1023 – 1KB). Esta será a região reservada na memória para transferência de dados entre nossos *streams*.

> **Nota**: Se preferir utilizar um *PChar* como *buffer*, é aconselhável escolher um tamanho até 61440 *bytes* (240Kb).

Com o *buffer* definido, podemos agora chamar o método *Read* do *stream*. Passamos para ele onde devem ser colocados os *bytes* lidos (*buffer*) e a quantidade máxima de *bytes* que gostaríamos de ler. O retorno do método é o número de *bytes* lidos. Então, este é número que passaremos para o método *Write*, pois queremos escrever exatamente o que foi lido e está no *buffer*. A quantidade de *bytes* lidos será *0* somente quando não houver mais nada para ler. Portanto, a cópia será feita com sucesso.

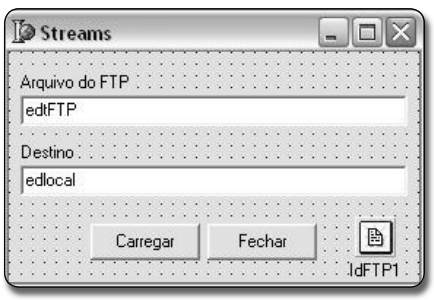


Figura 1. Exemplo de tela de download

Existem ainda alguns métodos não muito utilizados na classe *stream*. O método *seek* é utilizado para a movimentação do cursor a partir de um ponto (análogo ao método *MoveBy* do *TDataSet*). *ReadBuffer* e *WriteBuffer* utilizam *Read* e *Write*, respectivamente, e disparam uma exceção caso a quantidade de *bytes* a ser lida/escrita for diferente da recebida, ou se a quantidade de *bytes* a ser lida/escrita for igual a *0*. O restante dos métodos é utilizado pela IDE para a leitura/escrita de componentes e seus recursos em *streams*.

Das classes descendentes de *TStream* comumente utilizadas, podemos destacar: *TFileStream*, *TStringStream*, *TMemoryStream* e *TResourceStream*. As duas últimas serão demonstradas posteriormente e armazenam, respectivamente, *bytes* genericamente na memória e o conteúdo de um *resource*. Vale acrescentar que a classe *TFileStream* é bastante útil quando estamos lidando com uma quantidade muito grande de *bytes*, pois o armazenamento é feito diretamente no disco rígido, não na memória *RAM*. Operações no disco são, contudo, mais lentas do que na memória.

Uma última dica: muito cuidado ao misturar *TStringStreams* com outros *streams*. As *strings* do Delphi são *null-terminated*, ou seja, depois de todo o texto vem sempre um caractere nulo (*#0*). No caso de programas e arquivos não-texto, esse caractere é muito comum e não indica, de maneira alguma, o final do arquivo. Portanto, se você tentar copiar um *TFileStream* (carregando um arquivo binário) para um *TStringStream*, seu *DataString* apresentará os caracteres até apenas o caractere nulo (*#0*).

Com o que temos até agora já conseguimos fazer muita coisa no que se refere ao tratamento de arquivos (cópias, criação, leitura, modificação, etc...), no entanto, vamos praticar esse conhecimento através de um exemplo mais sólido.

Na internet, existem disponíveis diversos programas que aceleram a velocidade de *downloads*. Seu funcionamento básico compreende estabelecer várias conexões com o servidor e quebrar o *download* em pedaços. Posteriormente, o arquivo final é montado. Este é o exemplo que iremos construir.

Listagem 3. Trecho de código copiando a partir da posição 7

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileStreamIn: TFileStream;
  FileStreamOut: TFileStream;
  Buffer: array[0..1023] of Byte;
  BytesRead: Integer;
begin
  FileStreamIn := TFileStream.Create('caminho para arquivo de entrada',
    fmOpenRead);
  FileStreamOut := TFileStream.Create('caminho para arquivo de saída',
    fmCreate);
  try
    repeat
      BytesRead := FileStreamIn.Read(Buffer, 1024);
      FileStreamOut.Write(Buffer, BytesRead);
    until BytesRead = 0;
  finally
    FileStreamIn.Free;
    FileStreamOut.Free;
  end;
end;
```

Listagem 4. Criação da classe de download

```
TFTPThread = class(TThread)
private
  FLimit: Integer;
  FOrder: Integer;
  FTotal: Integer;
  FFileToGet: string;
  FIdFTP: TIdFTP;
  FStream: TStream;
  FFormToWarn: TForm1;
  procedure IdFTPWork(Sender: TObject; AWorkMode:
    TWorkMode; const AWorkCount: Integer);
protected
  procedure Execute; override;
public
  constructor Create(Order, Total: Integer;
    FileToGet: string; FormToWarn: TForm1);
  function GetStream: TStream;
end;
```

Criando um exemplo de download

Crie um novo projeto, salve-o como "FTPThread.dpr" e, no formulário principal, acrescente dois componentes *TEdit* (*edFTP* e *edLocal*) e um *TButton* (*btGet*). Vamos começar criando a classe da *Thread* que fará o *download* do arquivo. Declare abaixo da declaração do formulário uma classe como mostrado na **Listagem 4**. Desenhe uma tela semelhante a **Figura 1**.

Vamos precisar de: um atributo para saber qual é a ordem da *Thread* atual na lista de *Threads* que vão fazer parte

do *download* (*FOrder*); outro para saber a quantidade total de *Threads* (*FTotal*); esses dois serão utilizados para se calcular o limite de *bytes* que essa *Thread* fará o *download* (*FLimit*); o arquivo a ser baixado (*FFileToGet*); a *stream* na qual gravaremos o pedaço do arquivo (*FStream*); o componente que fará o *download* efetivamente (*TIdFTP*) e o formulário a ser avisado quando o *download* for concluído (*FFormToWarn*).

Não se esqueça de adicionar duas *Units* na sua cláusula *Uses*: *IdComponent* e *IdFTP*. Essas serão necessárias, pois faremos uso de um componente *TIdFTP* (que está na aba *Indy Clients*). A implementação do seu construtor será bem simples (**Listagem 5).**

Note que, como estamos passando *False* para o construtor da *Thread*, ela iniciará imediatamente, sem a necessidade de chamarmos seu método *Resume*. Implemente seu principal método conforme a **Listagem 6**.

Nossa primeira tarefa será criar e configurar o componente *TIdFTP*. Modifique as propriedades *Host*, *Username* e *Password* para as configurações de seu servidor FTP. O evento *OnWork* está sendo configurado para podermos finalizar o *download* quando o limite for atingido e será implementado a seguir. Conectamos então com o servidor e pegamos o tamanho do arquivo. O cálculo do limite será feito dividindo-se o tamanho do arquivo pelo número de *Threads*, sendo que a última *Thread* da lista ficará com o resto da divisão.

Internamente, o componente *TIdFTP* irá pegar a posição atual do *stream* de saída para saber de onde deve continuar o *download*. Portanto, criamos o *stream*, configuramos sua posição e iniciamos o *download*. Ao final, voltamos para a posição inicial e configuramos o *Size* do *stream*, pois o componente pode ter feito o *download* de alguns *bytes* a mais, que são desnecessários. Note que, agora estamos utilizando um *TMemoryStream*, pois o arquivo só será montado mais tarde. Você poderia utilizar aqui um *TFileStream* caso desejasse fazer uso de arquivos temporários para cada *Thread*. Por fim, chamamos o método *ThreadDone* do nosso formulário, que será implementado na seqüência. Implemente o tratador do evento *OnWork* do componente *TIdFTP* de acordo com adiante.

```
if AWorkMode = wmRead then
  if AWorkCount >= FLimit then
    FIdFTP.Abort;
```

Basicamente, estamos abortando a operação caso o limite de *bytes* tenha sido alcançado. Este evento é chamado cada vez que alguns *bytes* são carregados do servidor. Também pode ser utilizado no acompanhamento do progresso do *download*. Finalmente, como o formulário irá precisar montar o arquivo final, ele precisará dos *bytes* baixados por esta *Thread*, por isso precisamos do método *GetStream*, descrito a seguir.

```
function TFTPThread.GetStream: TStream;
begin
  Result := FStream;
end;
```

Nossa classe está pronta. Agora vamos à classe do formulário, que irá gerenciar as *Threads* e montar o arquivo final. Primeiramente, adicione antes da declaração da classe do formulário uma nova *class*.

```
  TFTPThread = class;
```

E na seção *private* da classe de seu formulário inclua:

```
  FThread1: TFTPThread;
  FThread2: TFTPThread;
  FThread3: TFTPThread;
  FThread4: TFTPThread;
  FThreadsDone: Integer;
....
```

Crie os objetos como mostrado a seguir:

```
procedure TForm1.btGetClick(Sender: TObject);
begin
  FThread1:=TFTPThread.Create(
    1, 4, edFTP.Text, Self);
 FThread2:=TFTPThread.Create(
    2, 4, edFTP.Text, Self);
 FThread3:=TFTPThread.Create(
    3, 4, edFTP.Text, Self);
 FThread4:=TFTPThread.Create(
    4, 4, edFTP.Text, Self);
end;
```

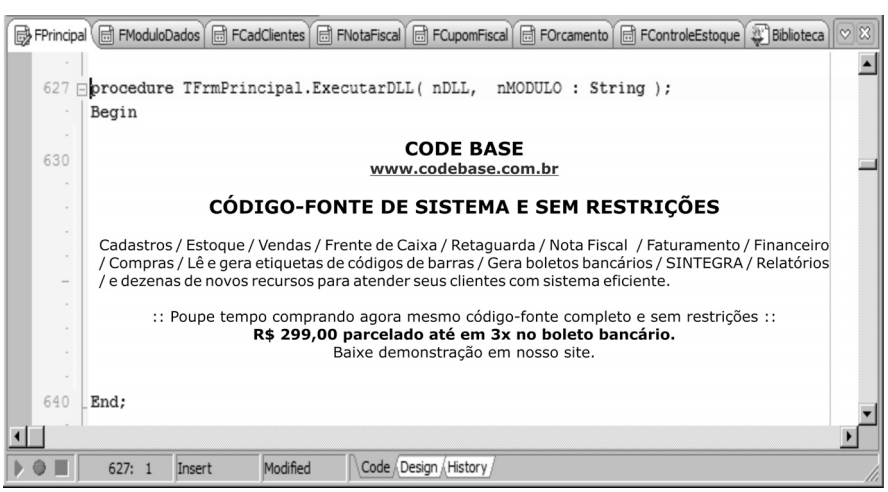
Finalmente, declare e implemente o método *ThreadDone* (**Listagem 7**).

Listagem 5. Código do método construtor

```
constructor TFTPThread.Create(Order,
  Total: Integer; FileToGet:
  string; FormToWarn: TForm1);
begin
  FOrder := Order;
  FTotal := Total;
  FFileToGet := FileToGet;
  FFormToWarn := FormToWarn;
  inherited Create(False);
end;
```

Listagem 6. Método principal da classe

```
procedure TFTPThread.Execute;
var
  ASize, AStart: Integer;
begin
  FIdFTP := TIdFTP.Create(nil);
  try
    FIdFTP.Host := 'localhost';
    FIdFTP.Username := 'Anonymous';
    FIdFTP.Password := '';
    FIdFTP.OnWork := IdFTPWork;
    FIdFTP.Connect;

    ASize := FIdFTP.Size(FFileToGet);

    FLimit := ASize div FTotal;
    AStart := FLimit * (FOrder - 1);
    if FOrder = FTotal then
      Inc(FLimit, ASize mod FTotal);

    FStream := TMemoryStream.Create();
    FStream.Position := AStart;
    FIdFTP.Get(FFileToGet, FStream, True);
    FStream.Size := AStart + FLimit;
    FStream.Position := AStart;

    Synchronize(FFormToWarn.ThreadDone);
  finally
    FIdFTP.Free;
  end;
end;
```



Listagem 7. Código do método ThreadDone

```
procedure TForm1.ThreadDone;
var
  AFileStream: TFileStream;
begin
  Inc(FThreadsDone);

  if FThreadsDone = 4 then
  begin
    AFileStream := TFileStream.Create(edlocal.Text, fmCreate);
    try
      Copy(FThread1.GetStream, AFileStream);
      Copy(FThread2.GetStream, AFileStream);
      Copy(FThread3.GetStream, AFileStream);
      Copy(FThread4.GetStream, AFileStream);
    finally
      AFileStream.Free;
    end;
  end;
end;
```

Listagem 8. Código do método Copy

```
procedure TForm1.Copy(const FromStream, ToStream: TStream);
var
  Buffer: array[0..255] of Byte;
  BytesRead: Integer;
begin
  repeat
    BytesRead := FromStream.Read(Buffer, 256);
    ToStream.Write(Buffer, BytesRead);
  until BytesRead = 0;

  FromStream.Free;
end;
```

Listagem 9. Código exemplo de compressão

```
procedure TForm1.Button1Click(Sender: TObject);
var
  StringStream: TStringStream;
  FileStreamOut: TFileStream;
  CompressionStream: TCompressionStream;
begin
  StringStream := TStringStream.Create('string no stream');
  FileStreamOut := TFileStream.Create('caminho para arquivo de saída', fmCreate);
  CompressionStream := TCompressionStream.Create(clDefault, FileStreamOut);

  try
    CompressionStream.CopyFrom(StringStream, StringStream.Size)
  finally
    StringStream.Free;
    CompressionStream.Free;
    FileStreamOut.Free;
  end;
end;
```

Listagem 10. Outra forma de compressão de arquivos

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FileStreamIn: TFileStream;
  StringStream: TStringStream;
  DecompressionStream: TDecompressionStream;
  ReadBytes: Integer;
  Buffer: array [0..1023] of Byte;
begin
  FileStreamIn := TFileStream.Create('caminho para arquivo de entrada', fmOpenRead);
  StringStream := TStringStream.Create('');
  DecompressionStream := TDecompressionStream.Create(FileStreamIn);

  try
    repeat
      ReadBytes := DecompressionStream.Read(Buffer, 1024);
      StringStream.Write(Buffer, ReadBytes);
    until ReadBytes = 0;
  finally
    DecompressionStream.Free;
    StringStream.Free;
    FileStreamIn.Free;
  end;
end;
```

Cada vez que uma *Thread* termina seu serviço, este método é chamado e quando as quatro terminarem, o arquivo é montado. A montagem do arquivo é feita normalmente, conforme os exemplos anteriores. Para isso utilizamos o método *Copy*. Então, declare e implemente-o. O código completo podemos ver na **Listagem 8**.

Neste caso, o método *CopyFrom* do *stream* não funcionaria porque precisamos manter as posições iniciais de cada *stream*, o que não acontece com o tal método.

Fica como sugestão de treino a criação de uma barra de progresso para o acompanhamento e o tratamento de alguns erros, como servidor indisponível, etc. Para a barra de progresso, verifique o quanto já foi carregado em cada *Thread* a cada *x* segundos. Dessa maneira, você pode também calcular a velocidade do seu *download* pegando a quantidade de *bytes* carregados nesse segundo.

Inicie um programa e faça o *download* de algum arquivo. Se tudo ocorrer bem, o arquivo final será montado corretamente e no local certo.

Compression com zLib

Certamente, quem nunca precisou, um dia vai precisar fazer a compactação de algo via programação. Para isso, muitos utilizam bibliotecas disponíveis na internet, acessam utilitários externos, etc. O que muita gente não sabe é que o Delphi já possui nativamente a biblioteca *zLib* para fazermos a compactação de arquivos. E sua utilização, com o conteúdo que temos até agora, é bastante simples. Utilizaremos, simplesmente, as classes *TCompressionStream* e *TDecompressionStream* da *unit zLib*. Analisemos o código da **Listagem 9**.

Conforme podemos perceber, a única mudança em relação ao primeiro exemplo foi a adição da classe *TCompressionStream*. Esta deve ser criada com base em um *stream* de saída (qualquer tipo) e devemos informar o nível de compressão desejado: *clNone*, *clFastest*, *clDefault*, *clMax*. De nenhum à compressão máxima, respectivamente. Ao chamarmos o método *CopyFrom* do *TCompressionStream*, ele irá ler os *bytes* do *stream* de entrada

e, à medida que a gravação é feita, a compressão é também realizada.

Vale lembrar aqui a importância do fechamento do *stream* de compressão (através do método *Free*). Caso contrário, o arquivo não será gravado corretamente. Além disso, tanto o *TCompressionStream* quanto seu *stream* de saída devem ser fechados corretamente. A leitura, nesse caso, seria o contrário. (**Listagem 10**)

Agora, em vez do *TCompressionStream*, estamos utilizando o *TDecompressionStream*. Sua operação, em vez de ser referente ao *stream* de saída, será no de entrada. E, mais uma diferença: a propriedade *Size*, neste caso, nos informa o tamanho do *stream* compactado. Se utilizarmos um *CopyStream* copiando esse número de *bytes*, não estaremos copiando todos os *bytes*, pois, o *stream* descompactado provavelmente será maior do que isso. Portanto, como não sabemos previamente o tamanho final, utilizamos os métodos *Read* e *Write* para conseguir ler todo o *stream*.

Como você deve ter reparado, essas classes não dão nenhum suporte para a adição de múltiplos arquivos em um mesmo *stream*. Temos que fazer isso manualmente. E é esse exemplo que construiremos a seguir. Vamos criar um programa similar ao *WinZip*, mas o padrão de compressão será nosso.

Primeiramente, vamos tentar entender como será o formato desse arquivo compactado. Para cada arquivo a ser adicionado no arquivo compactado, criaremos uma entrada. Ela terá a seguinte forma: [LENGTH DO NOME][NOME DO ARQUIVO][SIZE DO ARQUIVO][CONTEUDO DO ARQUIVO]. Assim, saberemos exatamente o que e quanto ler na hora de descompactar em múltiplos arquivos. O conteúdo do arquivo completo será uma entrada após a outra.

Para o *length* do nome do arquivo e para o *size* do arquivo, utilizaremos um tipo *Integer* e um tipo *Int64*, respectivamente. Para gravá-los, poderíamos facilmente utilizar um *IntToStr* e gravar sua representação *string*. Porém, 2097152 *bytes* (2MB), por exemplo, ocuparia sete casas. Em vez disso, vamos armazenar este número como ele é visto pelo computador (forma binária), o que ocuparia apenas três casas, pois número é representado por 32, 0 e 0. Isso vai nos fazer economizar um bocado de espaço no arquivo final.

Crie um novo projeto, salve-o como "Compression.dpr", inclua uma nova *Unit* e salve-a como "Compressor.pas". Nessa nova *Unit*, adicione *SysUtils*, *Classes*, *ZLib e Math* na sua seção *Uses* e declare o *Record* que irá representar uma entrada no arquivo e o tipo *TBuffer*, como podemos ver adiante.

```
TCompressionEntry = record
  Path: string;
  FileSize: Int64;
end;
TBuffer = array[0..1023] of Byte;
```

O tipo *TBuffer* é necessário para podermos passá-lo como parâmetro, o que não é permitido com *arrays*. Declare também uma classe *TCompressor*. Esta será a classe responsável pela compressão dos arquivos. Adicione a essa classe os seguintes métodos utilitários e implemente-os (**Listagem 11**).

Não se esqueça de adicionar a diretiva *overload* aos dois métodos *IntToBin*. Tais métodos não serão explicados a fundo porque são apenas utilitários e fogem um pouco ao contexto do exemplo, porém, serão extremamente úteis. Por exemplo, o método *CopyToBuffer* faz-se necessário pois, quando utilizamos o tradicional *StrPCopy*, caso haja algum caractere nulo no meio da seqüência, este será interpretado erroneamente como caractere final.

Listagem 11. Métodos da classe TCompressor

```
function TCompressor.CopyToBuffer(const From: string;
  Count: Integer): TBuffer;
var
  i: Integer;
begin
  for i := 1 to Count do
    Result[i - 1] := From[i];
end;
function TCompressor.CopyFromBuffer(const From: TBuffer;
  Count: Integer): string;
var
  i: Integer;
begin
  Result := '';
  for i := 0 to Count - 1 do
    Result := Result + From[i];
end;
function TCompressor.IntToBin(Int: Integer): string;
var
  i: Integer;
begin
  Result := '';
  for i := 3 downto 0 do
  begin
    Result := Result + Chr(Int shr (i * 8));
  end;
end;
function TCompressor.IntToBin(Int: Int64): string;
var
  i: Integer;
begin
  Result := '';
  for i := 7 downto 0 do
  begin
    Result := Result + Chr(Int shr (i * 8));
  end;
end;
function TCompressor.BinToInt(Bin: string): Integer;
var
  i: Integer;
begin
  Result := 0;
  for i := 0 to 3 do
  begin
    Result := Result or Ord(Bin[i + 1]) shl ((3 - i) * 8);
  end;
end;
function TCompressor.BinToInt64(Bin: string): Int64;
var
  i: Integer;
begin
  Result := 0;
  for i := 0 to 7 do
  begin
    Result := Result or Ord(Bin[i + 1]) shl ((7 - i) * 8);
  end;
end;
```

Listagem 12. Código do método CloseStreams

```
procedure TCompressor.CloseStreams;
begin
  FreeAndNil(FStreamWrite);
  FreeAndNil(FStreamRead);
  FreeAndNil(FInternalStream);
end;
```

Listagem 13. Construtor da classe TCompressor

```
constructor TCompressor.Create(const FilePath:
  string; CleanFile: Boolean);
begin
  FFilePath := FilePath;
  if CleanFile then
    RestartStream(fmCreate)
  else
    RestartStream(fmOpenRead);
  FStreamRead := TDecompressionStream.Create(
    FInternalStream);
  try
    PopulateEntries;
  finally
    CloseStreams;
  end;
end;
```

Listagem 14. Método PopulateEntries

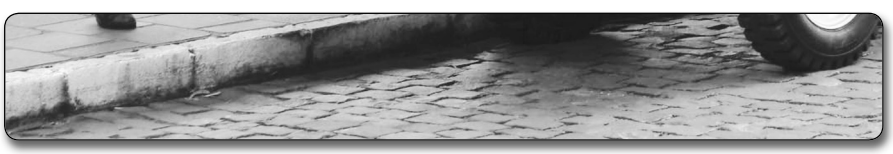
```
procedure TCompressor.PopulateEntries;
var
  Buffer: TBuffer;
  NameLength: Integer;
begin
  SetLength(FEntries, 0);

  while FStreamRead.Read(Buffer, 4) > 0 do
  begin
    SetLength(FEntries, Length(FEntries) + 1);
    NameLength := BinToInt(CopyFromBuffer(Buffer, 4));
    FStreamRead.Read(Buffer, NameLength);
    FEntries[High(FEntries)].Path := CopyFromBuffer(
      Buffer, NameLength);
    FStreamRead.Read(Buffer, 8);
    FEntries[High(FEntries)].FileSize := BinToInt64(
      CopyFromBuffer(Buffer, 8));
    ReadFile(FEntries[High(FEntries)].FileSize, False);
  end;
end;
```

Listagem 15. Método ReadFile

```
procedure TCompressor.ReadFile(Count: Int64;
  WriteResult: Boolean);
var
  Buffer: TBuffer;
  BytesRead: Integer;
begin
  repeat
    BytesRead := FStreamRead.Read(Buffer, Min(Count,
      1024));
    if WriteResult then
      FStreamWrite.Write(Buffer, BytesRead);
    Dec(Count, BytesRead);
  until (Count = 0) or (BytesRead = 0);
end;
```

Listagem 16. Método CopyEntireStream

```
procedure TCompressor.CopyEntireStream;
var
  Buffer: TBuffer;
  BytesRead: Integer;
begin
  repeat
    BytesRead := FStreamRead.Read(Buffer, 1024);
    FStreamWrite.Write(Buffer, BytesRead);
  until BytesRead = 0;
end;
```



Adicione *FInternalStream*, *FStreamWrite* e *FStreamRead*, os três do tipo *stream*, à declaração de sua classe. O primeiro referenciará o *stream* do arquivo final e, os outros dois serão os *streams* de compressão e descompressão, respectivamente. Declare e implemente o método como segue:

```
procedure TCompressor.RestartStream(Mode:
Word);
begin
  FInternalStream := TFileStream.Create(
    FFilePath, Mode);
end;
```

Este será utilizado para iniciar o *stream* principal e poderá ser sobrescrito em uma classe descendente para a gravação em outro tipo de *stream*. Adicione a diretiva *virtual* a ele. Aproveite para declarar e implementar o método *CloseStreams*, onde liberamos de memória todos os *streams* criados. Veja o código completo na **Listagem 12**.

Para as entradas, declare um atributo *FEntries* (*array of TCompressionEntry*) e adicione as propriedades:

```
    property EntriesCount: Integer read
GetEntriesCount;
    property Entries[Index: Integer]:
TCompressionEntry read GetEntry; default;
```

GetEntriesCount deve retornar simplesmente o *Length* de *FEntries* e *GetEntry* retorna *FEntries[Index]*. O construtor da classe deve ser declarado e implementado como na **Listagem 13**.

Declare *FFilePath* (*string*) para salvarmos o nome do arquivo. O parâmetro *CleanFile* irá definir se criaremos ou não um novo arquivo. Chamamos o recém criado *RestartStream* para inicializar o *stream* interno e criamos o *stream* de descompressão. Após isso, populamos o *Array* de entradas e fechamos os *streams*. Escolhemos por fechar os *streams* ao final de cada método para liberar o arquivo enquanto nenhuma operação está ocorrendo. O método *PopulateEntries* possui a implementação como na **Listagem 14**.

Sua implementação, por mais que pareça complicada, é, na verdade, bastante simples. Lemos quatro *bytes* (*Integer*) e armazenamos em *NameLength*. Lembrando que precisamos converter o valor lido de *TBuffer* para *string* (*CopyFromBuffer*) e, como este número estará na forma binária, precisamos restaurar o seu conteúdo

inteiro (*BinToInt*). Após isso, iremos ler o nome do arquivo e armazenamos na entrada. Finalmente, lemos oito *bytes* (*Int64*) para o tamanho do arquivo e, também o armazenamos na entrada. Como estamos lendo apenas a descrição de cada entrada e não seu conteúdo, chamamos *ReadFile* apenas para ler e descartar o conteúdo do arquivo, pois este não será necessário agora. Veja como fica nosso método *ReadFile* agora observando a **Listagem 15**, ou seja, simplesmente lemos o conteúdo do arquivo e, se necessário, gravamos no *stream* de saída.

Antes de iniciar a implementação da adição de um arquivo, vamos ao método *CopyEntireStream*. (**Listagem 16**). Este método copia todo o conteúdo do *stream* de leitura para o *stream* de escrita. Vamos então, finalmente, ao método *Add* previsto na **Listagem 17**.

Basicamente, iniciamos com a preparação dos *streams*. Note que estamos criando os *streams* de compressão e descompressão para o mesmo *stream* interno. Sim, isso é permitido. Após isso, copiamos todo o conteúdo atual de volta para o *stream*. Você deve estar se perguntando: mas não seria só uma questão de posicionar o *stream* de saída no final do arquivo? Pois a resposta é: não. O *stream* de compressão precisa ser único no arquivo inteiro. Se fecharmos ele, abrirmos o arquivo novamente, e continuarmos escrevendo, quando formos ler, o *stream* de descompressão pensará que o final do arquivo chegou quando fechamos o *stream* a primeira vez, ou seja, não irá funcionar.

Após isso, instanciamos um *stream* temporário para a leitura do arquivo a ser adicionado, inserimos uma entrada no *Array* e escrevemos o tamanho do nome, o nome e o tamanho do arquivo no *stream* de escrita. Finalmente, escrevemos o conteúdo do arquivo. Agora veja o método de extração de um arquivo (**Listagem 18**).

Listagem 17. Método Add da classe

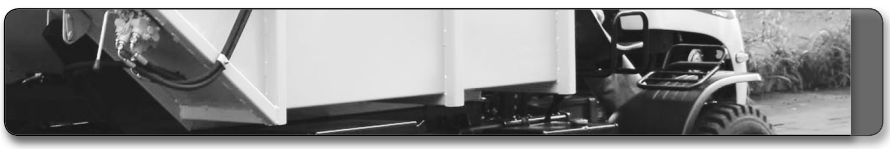
```
function TCompressor.Add(const FilePath: string):
  TCompressionEntry;
var
  Buffer: TBuffer;
  ATempStream: TFileStream;
  AFileName: string;
begin
  RestartStream(fmOpenReadWrite);
  FStreamRead := TDecompressionStream.Create(
    FInternalStream);
  FStreamWrite := TCompressionStream.Create(clMax,
    FInternalStream);
  CopyEntireStream;

  try
    ATempStream := TFileStream.Create(FilePath, fmOpenRead);
    try
      AFileName := ExtractFileName(FilePath);

      SetLength(FEntries, Length(FEntries) + 1);
      Result.Path := AFileName;
      Result.FileSize := ATempStream.Size;
      FEntries[High(FEntries)] := Result;

      Buffer := CopyToBuffer(IntToBin(Length(AFileName)), 4);
      FStreamWrite.Write(Buffer, 4);

      Buffer := CopyToBuffer(AFileName, Length(AFileName));
      FStreamWrite.Write(Buffer, Length(AFileName));

      Buffer := CopyToBuffer(IntToBin(ATempStream.Size), 8);
      FStreamWrite.Write(Buffer, 8);

      FStreamWrite.CopyFrom(ATempStream, ATempStream.Size);
    finally
      ATempStream.Free;
    end;
  finally
    CloseStreams;
  end;
end;
```

Listagem 18. Método de extração dos arquivos no stream

```
procedure TCompressor.Extract(Index: Integer; const
  ExtractTo: string);
var
  Buffer: TBuffer;
  ATempStream: TStream;
  Entry: TCompressionEntry;
  i: Integer;
begin
  RestartStream(fmOpenRead);
  FStreamRead := TDecompressionStream.Create(
    FInternalStream);
  try
    Entry := FEntries[Index];

    for i := 0 to Index - 1 do
      ReadEntry(False);

    FStreamRead.Read(Buffer, 4);
    FStreamRead.Read(Buffer, BinToInt(CopyFromBuffer(
      Buffer, 4)));
    FStreamRead.Read(Buffer, 8);

    ATempStream := GetDestStream(
      IncludeTrailingPathDelimiter(ExtractTo) +
      Entry.Path);
    try
      ATempStream.CopyFrom(FStreamRead, Entry.
        FileSize);
    finally
      CleanDestStream(ATempStream);
    end;
  finally
    CloseStreams;
  end;
end;
```

Conclusão

E com isso finalizamos a primeira parte deste artigo. Até a próxima edição, onde continuaremos a construção do exemplo. ●



Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Desenvolvendo uma Aplicação Completa com PocketStudio

Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 2

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atual como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

Continuando o artigo da edição anterior, iremos criar uma tela de consulta de clientes, e iniciar a manipulação de um banco de dados no Palm. Para iniciarmos, precisamos entender como funciona um banco de dados no PalmOS.

Banco de dados no PalmOS são estruturas completamente diferentes do que estamos acostumados a trabalhar na plataforma Windows. Internamente, os bancos de dados no Palm nada mais são que estruturas muito simples de registros, organizados seqüêncialmente sem conceito de campos, portanto não existe ordenação, chaves primárias, índices etc. Para facilitar nosso trabalho, o PocketStudio conta com a *Unit PsDatabase*, que faz parte da *PSLibrary*, com a qual é possível utilizar uma estrutura de campos nos registros.

Para criarmos um banco de dados no Palm devemos seguir os seguintes passos:

- Criar uma *Unit* para o Banco de Dados**:** todo banco tem uma estrutura própria de registros, além de configuração de *Name*, *CreatorID* e *DatabaseType*, devemos criar uma *Unit* para cada banco de dados;
- Abrir e Fechar o Banco de Dados: o local ideal para abrirmos e fecharmos os bancos de dados utilizados em nossa aplicação é na *Unit* principal de nosso projeto. Nas seções *StartApplication* e *StopApplication;*
- Criar funções de gravação, recuperação e validação dos dados: na *Unit* criada para o *BD* devemos criar uma função para cada procedimento anterior.

Todo banco de dados no Palm deve ter: um *Name***,** um *CreatorID* e um *DatabaseType*. O nome do banco de dados será utilizado para abrirmos e fecharmos o *BD* na aplicação. O *CreatorID* é um identificador de quatro dígitos que representa os campos da aplicação e o *DatabaseType* é um identificador único do conjunto de

dados dentro dos diversos bancos que uma aplicação pode ter. Também tem quatro dígitos. Devemos criar identificadores diferentes para cada banco de dados dentro de nossa aplicação.

O primeiro passo é adicionarmos uma nova *Unit* para montarmos a tabela de *Clientes*. Para isso vamos em *File|New>Unit*. Uma nova *Unit* é adicionada ao nosso projeto como mostra a **Figura 1**.

Vamos salvá-la com o nome de "ClientesDB.pas". Vale lembrar que podemos utilizar as teclas de atalhos *Ctrl + Shift + S* para salvar nosso projeto. Feito isso vamos montar nossa base de dados de clientes. Após criarmos a nova *Unit*, o próximo passo é codificarmos.

Logo abaixo de *Interface* vamos declarar uma seção *Uses* para adicionarmos a *PSL* que é a biblioteca do PocketStudio onde estão os métodos necessários para manipularmos a base de dados, dentro da *Unit PSDatabase*. Após a sessão *Uses* declaramos uma nova seção *Const* onde informaremos o *DBName*, *DBType* e os índices para recuperação e configuração dos valores nos campos.

Após a definição dos campos e seus respectivos índices, declaramos uma secão *Var*, onde informaremos ao PocketStudio o *FieldDefs* que é um *array* e possui o tipo dos campos. Também iremos declarar as variáveis globais que irão referenciar nossa base de dados em todo projeto. Veja na **Listagem 1** o código completo de criação da tabela.

> **Nota**: Muita atenção na definição dos tipos de dados dentro de um *array FieldDefs*, pois será a partir dessa definição que referenciamos o banco de dados em todo o projeto.

Na **Tabela 1** podemos ver os tipos básicos válidos no PocketStudio para uso em aplicações Palm.

Note que declarei duas variáveis *bClienteInclui* e *DBCli*. A primeira consiste em uma variável *boleana* usada para controlar a tabela e faremos uso dela para inserção/edição. A variável *DBCli* é responsável por fazer referência ao banco de dados aberto. Nas próximas funções entenderemos melhor seu

Figura 1. Unit do banco de clientes

Tipo	Descrição	Equivalente
Boolean	Condicional, armazena True/False	Boolean
UInt8	Inteiro, um byte, sem sinal. De 0..255	Byte
Int8	Inteiro, um byte, com sinal. De -127..128	ShortInt
UInt16	Inteiro, dois bytes, sem sinal. De 0..65535	Word
Int16	Inteiro, dois bytes, com sinal. De -32767..32768	SmallInt
UInt32	Inteiro, quatro bytes, sem sinal. De 0..4294967295	Cardinal
Int32	Inteiro, quatro bytes, com sinal. De -2147483648..2147483647	Integer
Double	Ponto flutuante, oito bytes. De 5.0 x 10^-327.. 1.7 x 10^308	Double
Single	Ponto flutuante, quatro bytes.	Single

Tabela 1. Tipos de dados no PalmOS

Listagem 1. Código de declaração da tabela Clientes

```
unit ClientesDB;
interface
uses PSL;
const
  { Nome do Banco de Dados }
  ClienteDBName = 'ClientesDB';

  { DatabaseType do Banco de Dados }
  ClienteDBType = Rsc('DBCL');

  { Índices para recuperação e configuração dos valores }
  { nos Campos do Banco de Dados de Clientes }
  Cli_Codigo = 0;
  Cli_Nome = 1;
  Cli_NomeFantas = 2;
  Cli_Cnpj = 3;
  Cli_Ie = 4;
  Cli_Fone = 5;
  Cli_Contato = 6;
  Cli_Endereco = 7;
  Cli_Cidade = 8;

var
  { Definição dos Campos }
  FieldDefs: array [0..8] of TFieldDef =
  (
    (DataType: ftUInt16),    // ClienteCodigo
    (DataType: ftString),    // ClienteNome
    (DataType: ftString),    // ClienteNomeFantas
    (DataType: ftString),    // ClienteCnpj
    (DataType: ftString),    // ClienteIe
    (DataType: ftString),    // ClienteFone
    (DataType: ftString),    // ClienteContato
    (DataType: ftString),    // ClienteEndereco
    (DataType: ftString)
  );

  { Variável para indicar se estamos incluindo/
    alterando no banco de dados }
  bClienteInclui: Boolean;

  { Variável que armazena a referência ao banco
    de dados aberto }
  DBCli : TDatabase;
```

funcionamento. A seguir criaremos três funções na base de dados. São elas:

- Open;
- Close;
- ProcuraCodigo;

Essas funções serão responsáveis por *abrir, fechar* e *realizar pesquisas* nos registros da base de dados quando necessário, respectivamente. Vejamos como declará-las.

Como já deve ter percebido, há basicamente duas áreas em nossas *Unit's* no PocketStudio. *Interface*, acima da palavra reservada *Implementation* e, claro, a própria área *Implementation*. Ambas funcionam exatamente como no Delphi. Uma (*Interface*) serve para declararmos variáveis, constantes, objetos, funções, procedimentos, tipos etc. A outra (*Implementation*) é onde ficam nossos códigos e a utilização de tudo que está declarado em *Interface*. Tendo isso em vista, declare as *function's* anteriormente citadas na área *Interface* conforme segue.

```
  function Open : Boolean;
  function Close : Boolean;
  function ProcuraCodigo(Codigo: UInt16):
Boolean;
```

Em seguida digite o corpo de cada função na área reservada as implementações. Veja na **Listagem 2** o código completo das declarações e funções.

Entendendo o código

Na primeira linha temos a chamada a seguinte função *Open* do objeto *PSDataBase*. O Primeiro parâmetro, *DBCli*, é a variável que acabamos de criar para referenciar a base de dados. O Segundo parâmetro, *ClienteDBName*, é a referência para o nome da tabela declarada no início da *Unit*. O terceiro e último, *dmModeReadWrite*, é muito importante. Ele indica que a base está sendo aberta para leitura e escrita.

Se necessário podemos passar o valor *dmModeReadOnly* indicando que o banco será aberto apenas para leitura. Caso o retorno dessa função seja falso, ou seja, não há banco de dados criado, então chamamos outro método, o *CreateDataBase* também do objeto *PSDataBase*.

Nesse caso estaremos criando um novo *Database*. Passamos para o método o nome da tabela (*ClienteDBName*), o identificador (*Creator*) e a variável de referência a tabela (*ClienteDBType*). O resultado da criação da tabela é passado para a variável *Result*. Caso o *Result* seja verdadeiro, então abrimos a tabela recém criada. Não havendo sucesso na abertura ou criação da tabela enviamos uma mensagem ao usuário informando o possível erro. O trecho a seguir demonstra como fazemos:

```
  if not Result then
 begin
    ShowSystemError(PSDatabase.LastError);
    exit;
  end;
```

O método *ShowSystemError* está declarado dentro da *Unit PSUtils* e irá retornar um erro baseado na exceção ocorrida. Para finalizarmos a função *Open*, chamamos o método *SetFieldDefs* para criação efetiva dos campos da tabela.

Nota do DevMan

É importante que o método SetFieldDefs seja chamado. Do contrário o compilador não acusará nenhum erro e teremos problemas ao distribuir a aplicação.

Em seguida podemos criar a função para fechar a tabela. O procedimento é bem simples, pois precisamos apenas chamarmos o método *Close* informando a variável *DBCli* para fechá-la. Portanto, digite o código a seguir:

```
  function Close : Boolean;
  begin
    Result := PSDatabase.Close(DBCli);
  end;
```

Listagem 2. Declaração de funções e seus respectivos códigos

```
..
implementation
{ Abre a base de dados, senão conseguir cria
  uma base nova }
function Open : Boolean;
begin
  { Tenta abrir o banco de dados }
  Result := PSDatabase.Open(DBCli, ClienteDBName,
    dmModeReadWrite);
  if not Result then
  begin
    { Se não conseguir, tenta criá-lo }
    Result := PSDatabase.CreateDatabase(ClienteDBName,
      Creator, ClienteDBType);
    if Result then
      { Depois de criado, abre o banco de dados }
      Result := PSDatabase.Open(DBCli, ClienteDBName,
        dmModeReadWrite);
  end;
  { Se não conseguiu abrir nem criar o banco de dados,
    retorna erro }
  if not Result then
  begin
    ShowSystemError(PSDatabase.LastError);
    exit;
  end;
  { Configura a estrutura do registro no banco de
    dados }
  PSDatabase.SetFieldDefs(DBCli, FieldDefs[0],
    SizeOf(FieldDefs) div SizeOf(FieldDefs[0]));
end;

function Close : Boolean;
begin
  Result := PSDatabase.Close(DBCli);
end;

function ProcuraCodigo(Codigo: UInt16): Boolean;
begin
  { Vamos varrer o banco do início ao fim procurando
    pelo código }
  PSDatabase.First(DBCli);
  while not PSDatabase.EOF(DBCli) do
  begin
    { Recuperamos o código para comparação }
    if PSDatabase.FieldUInt16(DBCli, Cli_Codigo)=
      Codigo then
    begin
      Result := True;
      exit;
    end;
    { Vamos ao próximo registro }
    PSDatabase.Next(DBCli);
  end;
  { Ok, não encontrou o código }
  Result := False;
end;
```

Vamos ao último método da nossa base de dados de *Clientes*, tão importante quanto abrir e fechar a base. Para termos a possibilidade de localizar um determinado registro, poderíamos montar essa função em uma *Unit* separada da tabela, mas para facilitar a manutenção de nossa aplicação colocaremos na mesma *Unit*.

O procedimento é bem simples e para quem já programa em Delphi é ainda mais fácil. Logo de início, enviamos o cursor para o início da tabela invocando o método *First*. Envolvemos a tabela em um laço *while* onde faremos a busca de informações. A cada interação do *loop* comparamos o valor do campo *Código* com o valor da variável *Codigo*, que recebemos no parâmetro da função. Caso seja positivo, significa que encontramos o dado no banco, então paramos o *loop* usando *exit*. Veja o trecho explicado:

```
if PSDatabase.FieldUInt16(
  DBCli, Cli_Codigo) = Codigo then
begin
  Result := True;
  exit;
end;
```

Perceba que o tipo de dado do parâmetro de nossa função é igual ao tipo de dado do campo, ou seja, *UInt16*. Isso se faz necessário para facilitar na hora de efetuar a pesquisa. E ambos os tipos devem ser os mesmos, caso contrário o método não encontrará o valor passado como parâmetro. Tão importante quanto o tipo do campo e do parâmetro é a variável que utilizamos para passar o valor para o parâmetro. Se utilizarmos uma variável de outro tipo tal como *UInt8* ou *UInt32*, o valor jamais será encontrado mesmo existindo. Sendo assim, muita atenção ao usar essa função.

Criando a tela de consulta a clientes

Após entendermos o funcionamento da tabela de clientes, vamos criar a tela para localizarmos os registros e exibirmos na janela de consulta. O primeiro passo é adicionarmos um novo formulário. No menu principal do PocketStudio selecione *File|New>Form* e salve-o como "UDadosClientes.pas" usando a opção *File>Save*. Feito isso altere a propriedade *Name* para "FrmDadosClientes" e *Title* para "Consulta a Clientes".

Na tela de consulta a clientes utilizaremos componentes que ainda não estudamos, e também trabalharemos com tipo *string*. Existem alguns conceitos a serem entendidos antes de iniciarmos. Abriremos um parênteses aqui para entendermos esses conceitos.

Entendendo strings

Não existe um tipo *string* no PocketStudio como existe em outras linguagens de programação. Para trabalharmos com *strings* precisamos utilizar a *API* do PalmOS. Há dois modos de declarar variáveis para manipulação de *strings*:

• Usar o tipo *PChar* que é um ponteiro para a área de memória onde a *string* será armazenada. Nesse caso, usaremos a *Unit PSString* da *PSL* para alocar e desalocar espaço em memória para manipulação do *Pchar*;

• Usar um *Array* de caracteres predefinidos. Nesse caso a variável não precisa de alocação nem desalocação, pois será criada e destruída pelo próprio sistema operacional, quando seu escopo for iniciado e terminado;

Muito cuidado com *strings*, pois o principal problema das aplicações está relacionado ao mau uso desse tipo. Os erros mais comuns são: esquecer de alocar ou alocar pouco espaço, ou ainda desalocar e liberar a memória.

Quando alocamos espaço para um tipo *PChar* precisamos lembrar de reservar um espaço a mais para o identificador de final de *string*, que no PalmOS é o caracter *ASCII #0* ("zero"). Não importa quanto tamanho reservamos para a *string*, quando o PalmOS encontrar o *#0* interpretatá como final de *string*.

Para manipularmos, isso utilizaremos a *Unit PSString* da *PSL*. Na **Tabela 2** podemos ver os principais tipos de *strings*.

Na **Tabela 2** estão as principais funções de manipulação de *strings* previstos em *PSL*. Antes de encerramos esse esclarecimento sobre *strings*, vale ressaltar novamente que praticamente 80% dos erros e problemas encontrados tanto no desenvolvimento quanto na implementação de aplicações desenvolvidas para PalmOS estão relacionados ao mau uso de *strings* como dito anteriormente. Muita atenção ao alocar memória para os tipos *PChar*. Se alocarmos pouco espaço, ou esquecermos de desalocar, um erro será gerado e em muitos casos o Palm é resetado revelando falha no sistema operacional e pode causar muitos danos ao dispositivo.

Trabalhando com campos

Os *Field's* (*TextBoxes*) estão na aba *Form*, ele é o segundo componente identificado pelo letras *abc*. Campos ("Fields") no PalmOs não tem a propriedade máscara, e não podemos formatar a entrada de dados. Para manipularmos esses componentes utilizamos a *Unit PSField* que faz parte da *PSL*. As principais funções da *PSField* são:

• *SetText*: Colocar o texto em um *Field*;

• *Text*: Recuperar o texto digitado em um *Field* e armazena em uma *string*;

• *Editable (Booleano):* Indica se o campo pode ser alterado;

• *MaxChars (Integer):* Indica a quantidade de caracteres aceitos;

• *MultipleLines (Booleano):* Indica se o campo terá multiplas linhas (simulando um memo);

• *Numeric (Booleano):* Indica se o campo

Função	Definição
StrCopy	Usamos para carregar um valor em uma String
StrCat	Usamos para concatenar duas Strings
StrIToA	Usamos para converter um valor Inteiro para Caracter
StrAToI	Usamos para converter um valor Caracter para Inteiro
FlpAToF	Usamos para converter um Caracter em um Ponto Flutuante
FlpFToA	Usamos para converter um Ponto Flutuante em um Caracter
PsString.Alloc	Usamos para alocar memória para um tipo PChar
PsString.Dispose	Usamos para liberar a memória para um tipo PChar
StrCompare	Usamos para comparar duas Strings
PsString.Dispose	Usamos para liberar a memória para um tipo PChar
StrCompare	Usamos para comparar duas Strings

Tabela 2. Funções de String no PocketStudio

Listagem 3. Código dos Bitmaps adicionados

```
const
  Bitmap1 = AutoID;
resource
  BITMAP Bitmap1 'Bitmap1.bmp' {DENSITY 72/108/144}
    {DIRECTCOLOR} {TRANSPARENCY 0 0}
    {'Bitmap2.bmp' ...};
const
  Bitmap2 = AutoID;
resource
  BITMAP Bitmap2 'Bitmap1.bmp' {DENSITY 72/108/144}
  {DIRECTCOLOR} {TRANSPARENCY 0 0}
  {'Bitmap2.bmp' ...};
```

Listagem 4. Código completos dos Bitmaps adicionados

```
const
  BmpInicio = AutoID;
  BmpAnterior = AutoID;
  BmpProximo = AutoID;
  BmpFim = AutoID;
resource
  BITMAP BmpInicio 'First.bmp';
  BITMAP BmpAnterior 'Previous.bmp';
  BITMAP BmpProximo 'Next.bmp';
  BITMAP BmpFim 'Last.bmp';
```

aceita apenas caracteres numéricos;

• *Underline(Set):* Indica se o campo será ou não uma linha solida ou pontilhada;

• *HasScrollBar (Booleano):* Indica se o campo está associado a uma *ScrollBar*;

• *Visible (Booleano):* Indica se o campo irá aparecer ou não em *runtime*;

• *DinamicSize (Booleano):* Indica se o campo será expandido automaticamente.

Trabalhando com Label

O objeto *Label* está na aba *Form* e é identificado pela letra "A". Seu funcionamento é praticamente idêntico ao *Label* de outras linguagens de programação, exceto por uma configuração na propriedade *Caption* via código que veremos mais adiante. Suas principais propriedade são:

• *Caption (String):* Indica o texto que será mostrado no *Caption*;

• *Font(Set):* Indica o tipo de fonte que será utilizada;

• *Left(Integer):* Indica o alinhamento a esquerda na tela;

• *Name(String):* É o *ResourceID* do *Label*;

• *Visible(Boolean):* Indica se o *Label* irá aparecer ou não em *runtime*.

Para manipularmos os componentes *Label*, utilizamos a *Unit PSLabel* que faz parte da *PSL*. As principais funções da *PSLabel* são:

• *SetCaption:* Responsável por atualizar a propriedade *Caption*;

• *Caption:* Recuperar um texto da propriedade *Caption*.

O que precisamos lembrar sobre as *Label's* é que devemos definir o espaço que será ocupado pelo *Caption* em *design*, e jamais usar mais espaço que o definido. Esta é uma limitação do PalmOS e não do PocketStudio.

Configurando os componentes da tela de consultas

A **Figura 2** mostra o layout da tela que criaremos. São ao todo nove *Label's* e *Field's*, todos da paleta *Form*. Insira também quatro *Button's* na parte inferior da tela e outro ao lado do *Caption* de novo formulário. Os botões receberão os nomes de "BtnInicio", "BtnAnterior", "BtnProximo", "BtnFim" e "BtmVoltar", respectivamente.

Os *Label's* receberão o respectivo *Caption* como mostrado na **Figura 2**. Como não faremos nenhuma alteração de propriedades em tempo de execução com os *Label's*, não se faz necessário alterar sua propriedade *Name*, poupando assim tempo de programação. Já os campos de texto, receberão valores em *runtime*, por isso faremos referência a eles o tempo todo. Pensando nisso fica mais fácil se alterarmos o *Name* de cada componente. Sendo assim, altere a propriedade *Name* de cada *Field* para "FldCodigo", "FldNome", "FldFantasia", "FldEndereco", "FldCidade", "FldCnpj", "FldIE", "FldFone" e "FldComprador", seguindo a ordem de componentes de cima para baixo.

Agora faremos a inserção dos *Bitmaps* em nosso projeto. No artigo anterior, aprendemos que para inserir menus devemos incluí-los e alterá-los diretamente no código-fonte da aplicação. Os *Bitmap's* funcionam da mesma forma. Pressione *F12* para visualizar o código e localize o início da seção *Implementation*. Selecione a paleta *Picture* e clique no componente *Bitmap*. Em seguida dê um clique abaixo da seção *Uses*. O PocketStudio fará a adição das seguintes instruções:

```
const
  Bitmap1 = AutoID;
resource
  BITMAP Bitmap1 'Bitmap1.bmp'
    {DENSITY 72/108/144}
    {DIRECTCOLOR} {TRANSPARENCY 0 0}
    {'Bitmap2.bmp' ...};
```

Não precisamos no momento de todas essas informações, portanto faremos pequenos ajustes. Cada *Bitmap* adicionado ao código faz com que o PocketStudio crie uma nova constante e uma nova área de *resource*, como podemos ver na **Listagem 3**.

Porém, nada impede que façamos o agrupamento dos *resources* e constantes para melhor organizar nosso código, ou ainda, podemos digitar diretamente no código a declaração dos *Bitmap's*. Veja na **Listagem 4** o código completo de todos os quatro *Bitmap's* adicionados. Note que o código está mais limpo e organizado. Retiramos os comentários de ajuda do PocketStudio.

Os *Bitmap's* são componentes não visuais e são inseridos diretamente no fonte da *Unit* como acabamos de fazer. As imagens a eles atribuídas devem ter extensão *.bmp*, tamanho de 15x9 e estarem localizadas no diretório do projeto. É necessário que você crie quatro imagens com os nomes "First.bmp", "Previous.bmp", "Next.bmp" e "Last.bmp". Perceba que os nomes das imagens estão declarados na seção resource dos *Bitmap's*.

Por último devemos ligar os *Bitmap's* criados aos seus respectivos botões. Para isso pressione novamente *F12* de forma que possa ver o formulário. Selecione o primeiro botão e na propriedade *Bitmap*, note que todas as imagens criadas/declaradas no código fonte aparecem. Informe a imagem correspondente a cada botão.

Codificando a aplicação

Antes de codificarmos os botões, precisamos criar uma função para carregar os dados na tela de consulta conforme navegamos entre os registros. Para isso utilizaremos as funções da *Unit PsDataBase*. Na área *Implementation*, declare

e codifique uma nova função chamada *CarregaDados*, como na **Listagem 5**. Não esqueça que para a *procedure* ser visualizada por outros formulários, como o principal por exemplo, é necessário que seja inclusa também na seção *Interface*, assim como em Delphi, ou seja, inclua a linha a seguir abaixo da *function HandleEvent* presente na *Unit*.

Entendendo o código

Na função *CarregaDados* utilizamos uma variável *Buffer* que é um *array* de *Char* ("caracteres"). Para auxiliar a transformação do código do cliente que é um *UInt16* para o tipo de dado *string* utilizamos a função *StrIToA*. Após isso recuperamos o valor do registro no campo código da tabela e transformamos em *string*, configuramos a propriedade *Text* do *Field* para receber o mesmo. Todas as operações são realizadas com o auxílio a *Unit PsField* e o método *SetText*.

O próximo passo é codificarmos os botões iniciando pelo botão de *Voltar*. Clique duas vezes no mesmo e seremos levados ao fonte da *Unit* no evento *Select*. Apenas inclua o código a serguir:

```
FrmGotoForm(FrmPrincipal);
```

> **Nota**: Assim como em Delphi quando referenciamos um outro formulário, devemos incluir a sua *Unit* ao *Uses* do formulário atual. Para isso adicione *UPrincipal* na seção *Uses* abaixo de *Implementation*.

Agora faremos a codificação de cada botão de nossa tela. Clique duas vezes em cada botão e digite o código de cada botão seguindo o esquema da **Listagem 6**.

Note que em todos os botões utilizam os métodos de navegação *First, Prior, Next* e *Last* da *PsDataBase* passando como parâmetro nossa variável de referência a tabela de clientes *DBCli*.

Para finalizar a codificação da tela de

Listagem 3. Código dos Bitmaps adicionados

```
const
  Bitmap1 = AutoID;
resource
  BITMAP Bitmap1 'Bitmap1.bmp' {DENSITY 72/108/144}
    {DIRECTCOLOR} {TRANSPARENCY 0 0}
    {'Bitmap2.bmp' ...};
const
  Bitmap2 = AutoID;
resource
  BITMAP Bitmap2 'Bitmap1.bmp' {DENSITY 72/108/144}
  {DIRECTCOLOR} {TRANSPARENCY 0 0}
  {'Bitmap2.bmp' ...};
```

Listagem 4. Código completos dos Bitmaps adicionados

```
const
  BmpInicio = AutoID;
  BmpAnterior = AutoID;
  BmpProximo = AutoID;
  BmpFim = AutoID;
resource
  BITMAP BmpInicio 'First.bmp';
  BITMAP BmpAnterior 'Previous.bmp';
  BITMAP BmpProximo 'Next.bmp';
  BITMAP BmpFim 'Last.bmp';
```

Material Didático em Português • Turmas Reduzidas • Instrutores Certificados
COMPLETA INFRA-ESTRUTURA • FÁCIL LOCALIZAÇÃO • COFEE-BRAKE • ESTACIONAMENTO CONVENIADO

Faça Tecnologia em Sistema de Informação na FMG: Formamos profissionais para as Maiores Empresas do Brasil

Contribuindo para o aperfeiçoamento profissional dos profissionaios de TI, a **FMG IT Learning**, através de parcerias oficiais, promove os Treinamentos **ITIL Foundations, CobiT, SOX, PMI e ISO.**

O objetivo destes programas de formação é dotar o gestor de conhecimentos implícitos na metodologia das melhores práticas de gestão, com conteúdo e estudos reais de caso. especificamente elaborada para a área de TI.

fmg IT Learning

SOLICITE INFORMAÇÕES SOBRE NOVAS TURMAS

rua general glicério, 45 • 3ºandar • cep: 09015-190 • santo andré, são paulo – brasil
Informações: 11 **4427.4687** | www.fmgnet.com.br | cursos@fmgnet.com.br

www.fmgnet.com.br

consulta de clientes vamos codificar o evento *OnOpen* do formulário para apresentarmos os dados da tabela. Selecione o formulário, vá até o *Object Inspector* e na aba *Events* localize o evento citado. Clique duas vezes nele e adicione o código a seguir:

```
CarregaDados;
```

Antes de encerramos a parte 2 do nosso artigo é importante lembrar que quando trabalhamos com tabelas precisamos controlar seu estado, ou seja, se está aberta ou fechada antes de qualquer operação. Para evitar problemas e erros aconselho colocar um controle na *Unit* principal do projeto. Abra a *Unit Vendas*, que equivale ao *Project>Source* do Delphi, e localize os métodos *StartApplication* e *StopApplication* para abrir e fechar as tabelas. A partir de agora toda tabela que criamos iremos adicionar os métodos da abrir e fechar na *Unit* principal do projeto nas funções *StartApplication* e *StopApplication*. No *StartApplication* abrimos e no *StopApplication* fechamos como mostra a **Listagem 7**.

A explicação é bem simples. No método *StopApplication* apenas adicionamos uma chamada ao método *Close* do *ClientesDB* para que a tabela seja fechada. Já em *StartApplication* verificamos se a tabela *Clientes* pode ser aberta. Caso o sistema não consiga abrir então enviamos uma mensagem ao usuário e impedimos a abertura do formulário. Por fim, abra o formulário principal, clique duas vezes no botão *Clientes* e inclua o código a seguir para que possamos ter acesso ao cadastro de clientes.

```
FrmGotoForm(FrmClientes);
```

Listagem 5. Código da função de carregamento dos dados

```
procedure CarregaDados;
var
  Buffer: Array[0..20] of Char;
begin
  { Se não tiver nenhum registro na tabela nada será feito }
  if PsDataBase.RecordCount(DbCli) = 0 then
    exit;
  StrIToA(Buffer, PsDataBase.FieldUInt16(DBCli, Cli_Codigo));
  PsField.SetText(FldCodigo, Buffer);
  PsField.SetText(FldNome, PsDatabase.FieldStringPtr(DBCli, CLi_Nome));
  PsField.SetText(FldFantasia, PsDatabase.FieldStringPtr(DBCli, CLi_NomeFantas));
  PsField.SetText(FldCnpj, PsDatabase.FieldStringPtr(DBCli, CLi_Cnpj));
  PsField.SetText(FldIe, PsDatabase.FieldStringPtr(DBCli, CLi_Ie));
  PsField.SetText(FldFone, PsDatabase.FieldStringPtr(DBCli, CLi_Fone));
  PsField.SetText(FldComprador, PsDatabase.FieldStringPtr(DBCli, CLi_Contato));
  PsField.SetText(FldEndereco, PsDatabase.FieldStringPtr(DBCli, CLi_Endereco));
  PsField.SetText(FldCidade, PsDatabase.FieldStringPtr(DBCli, CLi_Cidade));
end;
```

Listagem 6. Código de todos os botões da tela de clientes

```
procedure BtInicioSelect;
begin
  PsDataBase.First(DBCli);
  CarregaDados;
end;
procedure BtAnteriorSelect;
begin
  PsDataBase.Prior(DBCli);
  CarregaDados;
end;
procedure BtProximoSelect;
begin
  PsDataBase.Next(DBCli);
  CarregaDados;
end;
procedure BtFimSelect;
begin
  PsDataBase.Last(DBCli);
  CarregaDados;
end;
```

Listagem 7. Código dos métodos StopApplication e StartApplication

```
function StartApplication: Boolean;
begin
  { Abre o banco de dados de Clientes.
    Se não conseguir abrir, }
  { não executa a aplicação}
  if not ClientesDB.Open then
  begin
    ShowMessage('Banco de Clientes não encontrado!');
    Result := False;
    Exit;
  end;
  Result := PSApplication.CheckROMVersion($2003000);
  if not Result then
  begin
    FrmCustomAlert(AlertIncompatible, '2.0', nil,
      nil);
    Exit;
  end;
  FrmGotoForm(FrmPrincipal);
end;
procedure StopApplication;
begin
  ClientesDB.Close;
  FrmCloseAllForms;
end;
```

Conclusão

Vimos nesse artigo o conceito de banco de dados no PalmOS, como criar e manipular registros da base de dados, analisamos os tipos básicos que utilizaremos em nossa aplicação e conhecemos os componentes utilizados para montar a tela de consulta a clientes. No próximo artigo criaremos a tela de consulta a produtos. Até o próximo artigo, e bons códigos a todos! ●

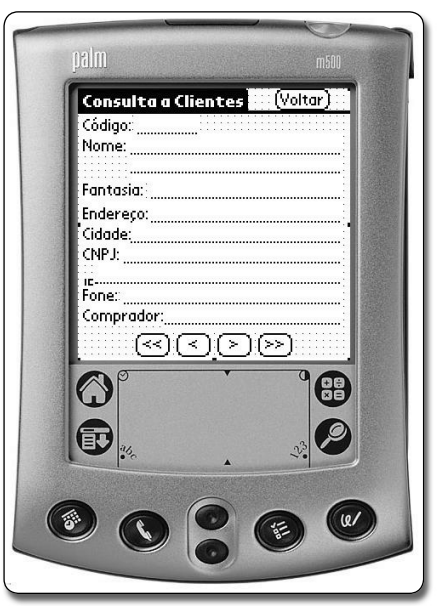


Figura 2. Exemplo de tela de cadastro

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir de **R$2,50!**

Saiba mais sobre o Sistema de Créditos!

DevMedia

www.devmedia.com.br/creditos

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Generics no RAD Studio 2007

Entendendo e utilizando Generics em suas aplicações .NET

Manoel Edesio B da Silva
(manoel.edesio@hkconsultoria.com.br)
detém as certificações em Borland C++ Builder, Borland Delphi for Win32 e Borland Delphi for .NET. Palestrou para a Borland em Delphi Meetings em alguns estados brasileiros, palestrante na Borland Conference de 2004, 2005, 2006 e 2007. Atualmente é diretor da HK Consultoria, empresa especializada em consultoria, treinamentos customizados e desenvolvimento de software em produtos Borland. Além dos quesitos anteriores a HK atua com projetos especializados em mobilidade com utilização de celulares e PDA'S.

Generics é um termo utilizado para se definir estruturas com tipos genéricos, para representar classes, métodos de classe etc; podendo os mesmos serem parametrizados com qualquer tipo de dado. O conceito do Generics já existe em linguagens como Java e C++. Em C++ o recurso é conhecido como *templates*. Todos nós já tivemos sérios problemas em escrita de código de forma genérica; principalmente todos os amantes de POO. Quando procuramos escrever um código portável às vezes nos deparamos com situações onde precisamos representar certo tipo de dado, porém esse tipo precisa ser modificado em *runtime* de acordo com a situação.

Algumas pessoas podem até achar que *generics* é algo parecido com *instanciamento* de classes descendentes em tipos ancestrais ou algo similar feito com *interfaces*, mas você poderá conferir em nosso artigo que esse maravilhoso recurso vai muito além do que o programador Delphi está acostumado. Ele irá simplesmente transformar a vida de todos os programadores em Delphi.

Esse recurso atualmente está disponível apenas para plataforma .NET. Já respondendo a pergunta de quando estará disponível para Win32, ainda não temos nada definido pela CodeGear. Mas já sabemos que em breve o recurso estará disponível para a nossa querida plataforma Win32.

Mais como assim? Qualquer tipo de dado?

Imagine que a partir de agora os tipos de dados de seus parâmetros, por exemplo, podem ser definidos em *runtime*, acredita? É verdade caros amigos *delphianos*, e é exatamente isso que nos dará a flexibilidade para fazer coisas absurdas em nossa linguagem.

Para ficar mais claro vamos criar uma nova aplicação utilizando o menu *File|New>VCL Forms Application – Del-*

phi for .NET no CodeGear RAD Studio 2007 e desenhar a interface da **Figura 1**. Salve o formulário como "uPrincipal.pas" e modifique seu *Name* para "frmPrincipal". O projeto salve como "Generics.dpr". Em seguida vamos adicionar uma nova *Unit* utilizando o menu *File|New>Other>Delphi for .NET Projects>New Files>Unit*, para definir a estrutura presente na **Listagem 1** e salve-a como "untPessoa.pas".

Agora imaginem o seguinte exemplo: iremos definir um *TList* para manipular objetos. Essa classe dá suporte a qualquer tipo de objeto em uma mesma lista, porém temos muitos problemas de desempenho através de conversões implícitas. Para recuperar as informações da lista precisamos sempre utilizar o recurso de *typecast* para manipular os objetos, além disso, se definirmos uma lista, por exemplo, para manipular objetos de uma classe chamada *TPessoa* não conseguimos garantir que serão inseridos apenas objetos derivados dessa classe. Com isso podemos inserir qualquer objeto em uma lista do tipo *TList* e como não temos restrição do tipo adicionado se fizermos alguma besteira, só vamos saber em *runtime* no momento de manipular o objeto através do *typecast*.

Para testar o exemplo vamos acessar o nosso formulário e dar um duplo clique no botão criar e adicionar o código da **Listagem 2**. Digitado o código vamos executar a aplicação pressionando *Shift+Ctrl+F9* e clicar no botão criar para testarmos o exemplo. Como estamos adicionando um *TButton* na lista e fazendo um *typecast* para *TPessoa* em

Listagem 1. Definição da classe TPessoa

```
unit untPessoa;
interface
  type
  TPessoa = class
  private
    FNome: string;
  public
    constructor Create(PNome:string);
    procedure setNome(const Value: string);
    property Nome:string read FNome write setNome;
  end;
implementation
constructor TPessoa.Create(PNome: string);
begin
  inherited Create;
  Nome := PNome;
end;
procedure TPessoa.setNome(const Value: string);
begin
  FNome := Value;
end;
end.
```

Listagem 2. Manipulação através de um TList (Click do botão criar)

```
procedure TfrmPrincipal.Button1Click(Sender: TObject);
var
  PersonList : TList;
  o : TObject;
  p : TPessoa;
begin
  PersonList := TList.Create;
  PersonList.Add(TPessoa.Create('Ana Beatriz'));
  PersonList.Add(TPessoa.Create('Pedro Henrique'));
  PersonList.Add(TPessoa.Create('Marilene Lima'));
  PersonList.Add(TPessoa.Create('Andreina'));
  { A linha a baixo provocará um erro em Runtime }
  PersonList.Add(button2);
  for o in PersonList do
  begin
    p := o as TPessoa;
    lstbxPessoas.Items.Add(p.Nome);
  end;
end;
```

Listagem 3. Manipulação de uma lista com Generics (Click do botão criar Generics)

```
procedure TfrmPrincipal.Button2Click(Sender: TObject);
var
  PersonList : List<TPessoa>;
  P : TPessoa;
begin
  PersonList := List<TPessoa>.Create;
  PersonList.Add(TPessoa.Create('Ana Beatriz'));
  PersonList.Add(TPessoa.Create('Pedro Henrique'));
  PersonList.Add(TPessoa.Create('Marilene Lima'));
  PersonList.Add(TPessoa.Create('Andreina'));
  { Erro em Tempo de Compilação visto que o tipo foi definido como TPessoa }
  PersonList.Add(button2);
  for p in PersonList do
  begin
    lstbxPessoas.Items.Add(p.Nome);
  end;
end;
```

Figura 1. Exemplo de tela para teste da classe TList

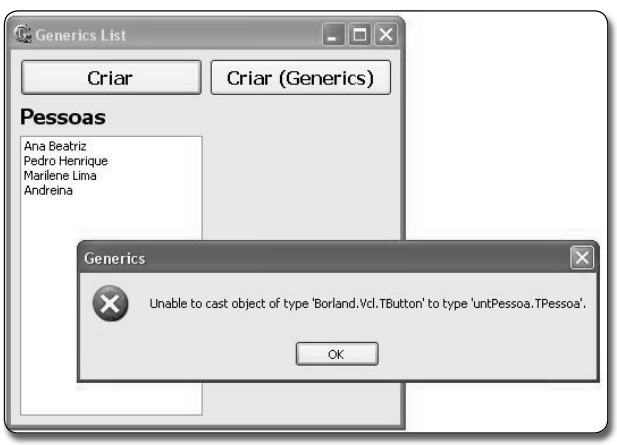


Figura 2. Exceção ocorrida em runtime ao fazer typecast do TButton para TPessoa em um TList

runtime teremos uma exceção de acordo com a **Figura 2**.

Primeiros passos

Considerando o exemplo anterior vamos implementar a mesma idéia através de *Generics*. De um duplo clique no botão *Criar(Generics)* e adicione o código da **Listagem 3**. Repare que no início do código temos o item *var* onde a definição do tipo genérico é feita entre "<" e ">". Como o tipo *List* dá suporte nativo a *Generics* só precisamos informar qual é o tipo que ele irá suportar. Em nosso código estamos dizendo com a definição *List <TPessoa>* que a lista é composta apenas por objetos do tipo *TPessoa*. A partir do momento da definição, a lista só irá aceitar objetos desse tipo; inclusive podemos observar isso no *Code Completion* do RAD Studio na **Figura 3**.

Ao tentar inserir um ***TButton*** ou qualquer outro objeto que não seja do tipo passado no ***Generic Type***, o compilador irá recusar o mesmo em tempo de compilação, não sendo necessário executar a aplicação para receber o erro em ***runtime***.

Além disso, podemos reparar que ao recuperar o objeto da lista não foi necessário fazer uso de *typecast*. Como o compilador reconhece o tipo automaticamente ele já tem todo o conteúdo de sua infra-estrutura, além de tudo temos o nosso querido *Code Completion* ao manipular os objetos da lista. É realmente incrível como o compilador realiza essa tarefa.

Um exemplo cotidiano

Um exemplo muito comum para demonstrar ainda o melhor o uso de *Generics* são as definições de *Pair Types*. Em nosso dia-a-dia sempre precisamos utilizar *pares combinados* de valores para manipulação através de classes. Porém em uma hierarquia de classes podemos ter muitas combinações em tipos de valores; imagine, por exemplo, uma combinação de dois valores. Podemos ter dois valores como *string*, um *string* e um inteiro, um inteiro e um real, um como *TEndereco* e outro como *TPessoa* etc.

O tipo utilizado sempre vai depender do momento da utilização em nossas aplicações. Já pensou se precisássemos ter em cada classe pares de valores? Certamente programando da forma tradicional teríamos que implementar essa particularidade em cada aplicação nossa, em cada classe etc.

Então qual seria a melhor maneira de resolver esse problema? A resposta é: criar um tipo genérico que aceite dois tipos diferentes em *runtime*.

Para demonstrar esse exemplo crie uma nova aplicação *VCL forms Application – Delphi for .NET* com no exemplo anterior e desenhe uma tela como na **Figura 4**. E em seguida adicione uma nova *Unit* e salve a mesma com o nome de "unitTypeGeneric.pas". Nessa *unit* vamos adicionar uma classe chamada *TPair* com a definição da **Listagem 4**.

Perceba que a classe é definida e como parâmetro ao tipo temos dois *generic types "TPair<TKey,TValue>"* que classificamos como *TKey* e *TValue*, ao fazer isso estamos oferecendo a possibilidade de que no momento da utilização do tipo *TPair* poderemos informar quais os tipos serão utilizados no lugar do *TKey* e no lugar do *TValue*. Em continuidade ao código criamos duas propriedades uma chamada *Key* e uma chamada *Value*. Veja que as propriedades são dos tipos genéricos definidos na classe. Com isso ao se definir um tipo genérico, se quisermos manipular o mesmo devemos sempre referenciá-los por meio dos nomes.

```
  PersonList.Add(TPessoa.Create('Ana Beatriz'));
  PersonList.Add(TPessoa.Create('Pedro Henrique'));
  PersonList.Add(TPessoa.Create('Marilene Lima'));
  PersonList.Add(TPessoa.Create('Andreina'));
//  PersonList.A item: TPessoa 2); //Erro em Tempo de Compilação
```

Figura 3. Code Completion já reconhece o tipo definido

Listagem 4. Definição da classe TPair

```
unit unitTypeGeneric;

interface
  type
    TPair<TKey,TValue> = class(TObject)
    private
      FKey: TKey;
      FValue: TValue;
      function GetKey: TKey;
      function GetValue: TValue;
      procedure SetKey(const Value: TKey);
      procedure SetValue(const Value: TValue);
    published
      property Key: TKey read GetKey write SetKey;
      property Value: TValue read GetValue write SetValue;
    end;

implementation

function TPair<TKey, TValue>.GetKey: TKey;
begin
  result := FKey;
end;

function TPair<TKey, TValue>.GetValue: TValue;
begin
   result := FValue;
end;

procedure TPair<TKey, TValue>.SetKey(const Value: TKey);
begin
  FKey := Value;
end;

procedure TPair<TKey, TValue>.SetValue(const Value: TValue);
begin
  FValue := Value;
end;
```

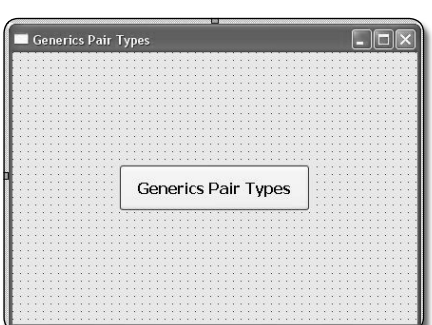


Figura 4. Interface da aplicação de exemplo Pair Types

Para utilizarmos agora a classe criada, vamos acessar a *unit* do nosso formulário principal e na seção *Type* antes da definição do *TForm* vamos adicionar a definição conforme a seguir:

```
type
  TSSPairOne = TPair<string,string>;
  TSSPairTwo = TPair<integer,real>;
```

Além disso, não podemos esquecer de adicionar na seção *interface* da *Unit* do formulário a referência para a *Unit unitTypeGeneric* na cláusula *Uses*. Esse código está definindo dois tipos, um deles contém um par de chaves onde o *Key* e *Value* são do tipo *string*, já o segundo define o *Key* como *Integer* e o *Value* como *Real*. Agora para executarmos um teste utilizando os nossos tipos dê um duplo clique no botão *Generics* e adicione o código da **Listagem 5**.

Nesse código temos a definição de duas variáveis, uma que utiliza o tipo *TSSPairOne* e outra que faz uso do *TSSPairTwo*, na seqüência vamos utilizar as variáveis definidas. Instanciamos cada uma delas com a classe equivalente passando como parâmetro o tipo definido e em seguida preenchemos as propriedades com valores que obrigatoriamente tem que seguir a definição de cada tipo. Observe na **Figura 5** que o tipo de cada propriedade já é identificado pelo RAD Studio no *Code Completion*, e se for passado um tipo diferente da definição o próprio compilador já avisa ao programador gerando um erro no código.

Trabalhando com Constraints

Já aprendemos como se definir um *Generic Type* e utilizar o mesmo informando o tipo suportado em *runtime*, porém da forma vista até agora o parâmetro sempre suportará qualquer tipo. Existe um recurso do *Generics* denominado *Contraints*, que é uma forma de gerar uma obrigatoriedade quanto ao tipo passado para um *Generics Type*. Esse recurso poderá ser aplicado a todas as formas de parametrização que inclui: classes, *records*, *interfaces* e ainda métodos parametrizados.

Para exemplificar o recurso vamos construir um exemplo onde o parâmetro passado ao *Generic Type* terá a obrigatoriedade de ser especificamente de um

```
  FSSPairTwo := TPair<System.int32,System.Double>.Create;
  FSSPairTwo.Key      := 1;
  FSSPairTwo.
end;

end.
```

procedure	**SetValue**[const Value: Double];
property	**Key**: Integer;
property	**Value**: Double;
constructor	**Create**;
function	**ToString**: string;

Figura 5. Code Completion exibindo os tipos definidos para as propriedades Key e Value

Listagem 5. Implementando o Pair Types (Click do botão criar Generics)

```
procedure TfrmPrincipal.Button1Click(Sender: TObject);
var
  FSSPairOne: TSSPairOne;
  FSSPairTwo: TSSPairTwo;
begin
  FSSPairOne := TPair <System.string, System.string>
    .Create;
  FSSPairOne.Key := 'Idade';
  FSSPairOne.Value := '10';
  FSSPairTwo := TPair <System.int32, System.Double>
    .Create;
  FSSPairTwo.Key := 1;
  FSSPairTwo.Value := 4.5;
end;
```

Listagem 6. Implementação do código da unit unitDefineGenericType

```
unit unitDefineGenericType;

interface

uses Dialogs;

type
  IComunicate = interface
    procedure Speak;
  end;
  TMammalCom < T: IComunicate > = class(TObject)
  private
    FMammal: T;
  public
    procedure DoSpeak;
    property Mammal: T read FMammal write FMammal;
  end;

  TMammal = class(TObject, IComunicate)
  private
    FName: string;
  public
    procedure Speak;
    property Name: string read FName write FName;
  end;

  TAndroid = class
  end;

implementation

procedure TMammalCom < T > .DoSpeak;
begin
  Self.Mammal.Speak;
end;

procedure TMammal.Speak;
begin
  ShowMessage('Mammal Falando!!!');
end;
end.
```

Figura 6. Evento clique do botão Generics Constraints

Listagem 7. Implementação do código da unit unitGenericTypeConstructor

```
unit unitGenericTypeConstructor;

interface

uses Dialogs;

type

  TNeedsConstructorGenericType < TypeConstructor: constructor > = class
  private
    FDados: TypeConstructor;
  public
    constructor Create;
    destructor Destroy; override;
    function GetData: TypeConstructor;
    procedure SetData(Value: TypeConstructor);
    property Dados: TypeConstructor read GetData write SetData;
  end;

  TRecordNoConstructor = record
    SomeInteger: integer;
  end;

  TClassHasConstructor = class
  public
    SomeInteger: integer;
    constructor Create;
  end;

implementation

constructor TNeedsConstructorGenericType < TypeConstructor > .Create;
begin
  inherited;
  Dados := TypeConstructor.Create;
end;

destructor TNeedsConstructorGenericType <TypeConstructor>.Destroy;
begin
  inherited;
end;

function TNeedsConstructorGenericType <TypeConstructor>.GetData: TypeConstructor;
begin
  result := FDados;
end;

procedure TNeedsConstructorGenericType <TypeConstructor>.SetData(Value:
TypeConstructor);
begin
  FDados := Value;
end;

constructor TClassHasConstructor.Create;
begin
  inherited;
end;

end.
```

tipo, e com isso vamos notar que ao manipular esse *Generic Type* teremos que obedecer essa regra, caso contrário não vamos conseguir utilizar o tipo criado.

Vamos lá mãos na massa agora! Crie uma nova aplicação *VCL forms Application – Delphi for .NET* utilizando o mesmo esquema dos exemplos anteriores. Apenas insira um TButton ao formulário principal e salve-o como "uPrincipal.pas". Em seguida vamos criar uma nova *Unit* chamada "unitDefineGenericType.pas" com a estrutura da **Listagem 6.** Dentro da *Unit* definimos individualmente uma *interface* chamada *IComunicate* com um método definido chamado *Speak*, na seqüência temos a definição de uma classe chamada *TMammalCom* que define um tipo genérico. Mais percebam que temos uma definição por *Constraint* na classe *TMammalCom<T: IComunicate>*. Esse *Constraint* está informando que o tipo que será definido ao *Generic Type* obrigatoriamente terá que implementar a *interface IComunicate*. Em seguida definimos duas classes, uma que implementa a interface e outra que não da suporte a mesma.

Após definição da *Unit* no formulário principal vamos dar um duplo clique no botão *Generic Constraints* e implementar o código a seguir**.**

```
procedure TForm1.Button1Click(Sender:
TObject);
 var
  MyMammal: TMammalCom < TMammal > ;
  MeuAndroid: TMammalCom < TAndroid > ;
begin
end;
```

Você poderá reparar que na primeira linha definimos uma variável para o tipo genérico passando como parâmetro a classe *TMammal* que dá suporte a *interface* e na seqüência definimos uma segunda variável passando a classe *TAndroid* que não tem suporte a *interface*. Ao compilar podemos observar ainda na mesma figura o erro gerado pelo compilador informando que o tipo *TAndroid* não dá suporte à *interface* (**Figura 6**).

Constraints com construtor

Continuando o exemplo de *Constraints* vamos exemplificar o tipo parâmetro *Constructor* na definição de classes. Com esse tipo de *constraints* vamos exigir que

na passagem de parâmetro do tipo seja obrigatoriamente exigido um construtor público. Caso contrário o compilador irá nos informar que o parâmetro não é válido. Agora repita os passos anteriores e crie um novo projeto *VCL forms Application – Delphi for .NET* inserindo apenas um botão ao formulário principal. Crie uma nova *Unit* com o nome "unitGenericTypeConstructor.pas" e defina seu código como na **Listagem 7**.

Estamos definindo nessa *Unit* uma classe chamada *TNeedsConstructorGenericType* que define um *Generic Type* com um constraint *constructor*, em seguinda temos a definição de uma classe e de um *record*. E a nossa classe tem um método construtor definido com visibilidade pública. Em seguida vamos até o formulário de nossa aplicação e dar um duplo clique no botão *Generic Type Constructor* e implementar o código da **Listagem 8**.

Estamos definindo duas variáveis baseados no tipo *TNeedsConstructorGenericType* porém um tipo passado como parâmetro é um *record* e o outro é uma classe, você poderá reparar que ao tentar compilar o projeto vamos receber do compilador do RAD Studio a mensagem de erro da **Figura 7** onde estamos sendo informados que o tipo parâmetro *TypeConstructor* precisa de um construtor público. Para executar a aplicação comente a linha que contém a definição *MyNoConstructorType:TNeedsConstructorGenericType<TRecordNoConstructor>;* e rode a aplicação.

Figura 7. Erro gerado pelo Generic Type Constructor

Listagem 8. Click do botão Generic Type Constructor

```
procedure TfrmPrincipal.Button1Click(Sender: TObject);
var
  MyConstructorType: TNeedsConstructorGenericType
    <TClassHasConstructor>;
  MyNoConstructorType: TNeedsConstructorGenericType
    <TRecordNoConstructor>;
begin
  MyConstructorType := TNeedsConstructorGenericType
    <TClassHasConstructor>.Create;
  try
    MyConstructorType.Dados.SomeInteger := 44;
    ShowMessage('O Valor é: ' + MyConstructorType
      .Dados.SomeInteger.ToString);
  finally
    MyConstructorType.Free;
  end;
end;
```

Conclusão

Por meio desse artigo podemos concluir que cada dia que passa estão sendo incluidos novos recursos na linguagem para facilitar o nosso dia-a-dia. O *generics* é um recurso utilizado junto a programação orientado a objetos que consegue reduzir drásticamente o esforço de programação quando temos a necessidade de manipular tipos que possam se alternar em determinados momentos. Espero ter contribuido através desse artigo para um melhor esclarecimento sobre esse interessante assunto e já incentivado a todos delphianos a inclusão do recurso em seus projetos .NET. ●

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Controle on-line de vídeo-locadora - Parte 1

Veja como criar um sistema on-line de controle para uma vídeo-locadora

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Junto com o aumento acelerado do uso da internet, a cada dia que passa a necessidade de empresas de todos os ramos e portes em ingressar com seu negócio no mundo virtual tem se tornado uma rotina. Surgem necessidades para as empresas encontrarem maneiras de negociar com o seu cliente à distância, ou seja, através da internet, sejam por meio de grandes portais de informações ou através de simples sistemas rodando na *Web* onde clientes/usuários credenciados podem interagir e realizar reservas, compras, vendas, entre outros.

O objetivo deste artigo é justamente a criação de um sistema *on-line*, onde a situação descrita será aplicada para as atividades diárias de uma vídeo-locadora. Veremos nesta e na próxima edição um passo-a-passo do desenvolvimento do aplicativo, que será dividido em duas formas de interação com usuários: administradores e usuários comuns. Onde toda a parte administrativa de manutenção de cadastros, reservas e retiradas de vídeos e um espaço para o cliente o qual com uso de seu usuário e senha poderá fazer a consulta de filmes disponíveis e realizar pela *Web* a reserva dos mesmos, além do acesso a um histórico de retiradas já efetuadas, economizando assim o tempo gasto na locadora para a busca do filme ideal.

A aplicação será desenvolvida sob a plataforma .NET Framework 1.1 com uso da ferramenta de desenvolvimento Borland Developer Studio 2006 for ASP.NET e banco de dados Firebird 1.5.

Nesta primeira etapa do artigo veremos a análise e criação da base de dados, criação da área de cadastros de filmes, gêneros, clientes e usuários, manutenção de reservas realizadas por clientes e novas reservas pelo administrador da locadora.

Antes de começar...

Antes de codificar qualquer coisa em ASP.NET, é importante que você tenha em mente alguns conceitos e tome al-

guns cuidados importantes, principalmente se está migrando de uma solução Desktop/VCL:

• Não queira fazer na Web as coisas exatamente como você faz em aplicações Windows/Desktop - o uso de *Web Forms* do ASP.NET ajuda, claro, mas saiba que você está rompendo de uma vez duas barreiras: mudança de framework (VCL para .NET FCL) e mudança de paradigma (Windows para Web). Um browser não é um formulário Delphi, infelizmente, então se acostume às limitações do HTML e HTTP;

• Saiba como funciona o modelo de execução do ASP.NET – é importantíssimo lembrar que o ASP.NET (e aplicações *Web* como um todo) são *stateless*: cada requisição é processada como se fosse a primeira. Além disso, o ASP.NET cria e recria o formulário no servidor para cada requisição, o que é bem diferente do modelo "TForm-based". É claro, temos alguns recursos para tratar sessão, cache, *ViewState* etc.

Começaremos então nosso exemplo, partindo pela conexão com o banco de dados.

Escolhendo o provider para ADO.NET

Temos somente uma opção para acessar o Firebird a partir de aplicações construídas para o .NET Framework: o ADO.NET. Sabendo disso, o próximo passo é escolher um *Provider* para ADO.NET que permita o melhor acesso ao banco de dados.

O .NET Framework 1.1 já é distribuído com quatro *Providers* nativos para ADO.NET:

SQL Provider - para acesso ao SQL Server;

Oracle Provider - para acesso ao Oracle;

OleDB Provider - para acesso a fontes de dados que possuam um driver OleDB;

ODBC Provider - para acesso a fontes de dados que possuam um driver ODBC;

Para acessar o Firebird, poderíamos usar qualquer um dos dois últimos *Providers* (ODBC ou OLEDB). No entanto, seria necessário instalar uma camada extra (um driver OLEDB ou ODBC), o que comprometeria o tempo de resposta, que é crítico em soluções *Web*.

Temos ainda mais duas opções:

BDP - Borland Data Provider - um *Provider* para ADO.NET construído pela Borland, distribuído com o Delphi.

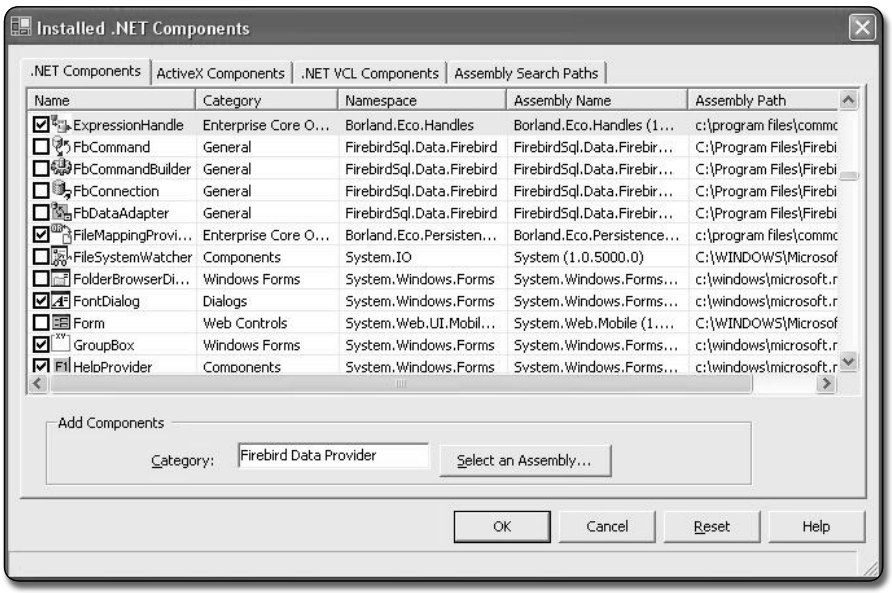


Figura 1. Instalação do provider .NET no Delphi

Firebird Data Provider – Provider para ADO.NET específico para o Firebird, desenvolvido e distribuído no mesmo site do banco de dados.

Se você desejar usar o BDP, deverá instalar um *driver* extra para o BDP, já que o Delphi suporta somente o acesso nativo ao InterBase, que como sabemos não é totalmente compatível com o Firebird. Nesse caso, optaremos pelo *Firebird Data Provider*, mais otimizado e específico para acessar o Firebird.

Download e instalação do Firebird Data Provider

Acesse o endereço *www.firebirdsql.com*, clique no link *Download* e a seguir em *Firebird .NET Data Provider*. Baixe e instale a última versão do *Data Provider for .NET Framework 1.1*, bastando seguir os passos no assistente que será iniciado.

> **Nota**: Neste artigo usaremos versão 1.7.1 do *Provider*, as demais versões estão disponíveis para Framework acima do 1.1 usado pelo BDS 2006.

Após a instalação, no Delphi 2006 clique no menu *Component|Installed .NET Components*. Digite "Firebird Data Provider" na opção *Category*, clique no botão *Select an Assembly* **(Figura 1)** e escolha o arquivo *FirebirdSql.Data.Firebird.dll*, localizado no diretório de instalação do *Provider*, por padrão em *C:\Arquivos de programas\FirebirdNETProvider(versão)*. Clique em *Ok* e observe que os novos componentes para acesso ao Firebird estão agora disponíveis no IDE **(Figura 2)**.

Figura 2. Componentes do provider .NET Firebird já instalados

Criando o banco de dados

Para iniciar o projeto, vamos primeiramente criar o banco de dados e as tabelas necessárias para cada tela de cadastro. As tabelas são: *Clientes*, *Usuarios*, *Generos*, *Locacao* e *Videos*. Neste artigo, criaremos o banco de dados utilizando a ferramenta IBExpert em sua versão *Standard*. Por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique

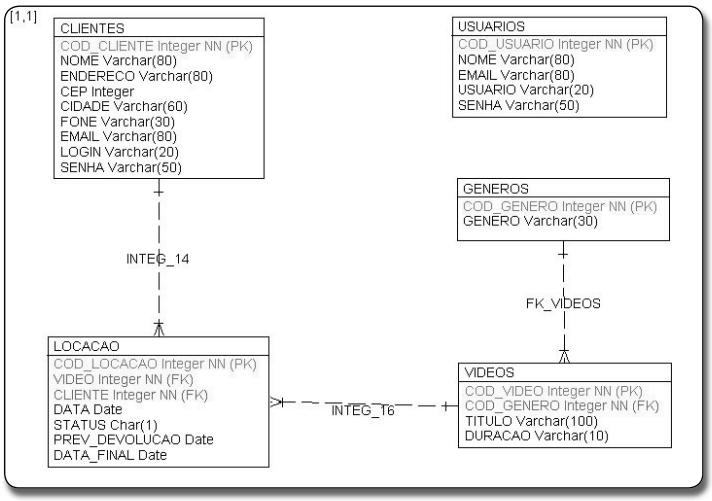


Figura 3. Modelo ER da base de dados

em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de *download* da ferramenta.

Nosso banco de dados será como mostrado no modelo ER visualizado na **Figura 3**. Criaremos as tabelas necessárias e seus respectivos índices e chaves estrangeiras conforme o modelo. Para agilizar o processo faremos a criação utilizando o recurso de execução de *scripts* no IBExpert. Abra então a ferramenta e vamos criar o banco de dados usando a opção *Script Executive* presente no menu *Tools*.

Com a tela de *scripts* aberta, digite o código da **Listagem 1**. Nele estão contidos os esquemas para a criação do banco de dados assim como de cada tabela do sistema.

Listagem 1. Script de criação da base de dados

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

SET CLIENTLIB 'C:\Program Files\Firebird\Firebird_1_5
  \bin\fbclient.dll';

CREATE DATABASE '<Caminho>\Locadora.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET WIN1252;

CREATE GENERATOR GEN_CLIENTES_ID;
SET GENERATOR GEN_CLIENTES_ID TO 1;
CREATE GENERATOR GEN_GENEROS_ID;
SET GENERATOR GEN_GENEROS_ID TO 6;
CREATE GENERATOR GEN_LOCACAO_ID;
SET GENERATOR GEN_LOCACAO_ID TO 40;
CREATE GENERATOR GEN_USUARIOS_ID;
SET GENERATOR GEN_USUARIOS_ID TO 0;
CREATE GENERATOR GEN_VIDEOS_ID;
SET GENERATOR GEN_VIDEOS_ID TO 7;

CREATE TABLE CLIENTES (
    COD_CLIENTE  INTEGER NOT NULL,
    NOME         VARCHAR(80) CHARACTER SET NONE,
    ENDERECO     VARCHAR(80) CHARACTER SET NONE,
    CEP          INTEGER,
    CIDADE       VARCHAR(60) CHARACTER SET NONE,
    FONE         VARCHAR(30) CHARACTER SET NONE,
    EMAIL        VARCHAR(80) CHARACTER SET NONE,
    LOGIN        VARCHAR(20),
    SENHA        VARCHAR(50)
);
CREATE TABLE GENEROS (
    COD_GENERO  INTEGER NOT NULL,
    GENERO      VARCHAR(30) CHARACTER SET NONE
);
CREATE TABLE LOCACAO (
    COD_LOCACAO     INTEGER NOT NULL,
    VIDEO           INTEGER NOT NULL,
    CLIENTE         INTEGER NOT NULL,
    DATA            DATE Default 'TODAY',
    STATUS          CHAR(1) CHARACTER SET NONE,
    PREV_DEVOLUCAO  DATE Default 'TODAY',
    DATA_FINAL      DATE
);
CREATE TABLE USUARIOS (
    COD_USUARIO  INTEGER NOT NULL,
    NOME         VARCHAR(80),
    EMAIL        VARCHAR(80),
    USUARIO      VARCHAR(20),
    SENHA        VARCHAR(50)
);
CREATE TABLE VIDEOS (
    COD_VIDEO   INTEGER NOT NULL,
    COD_GENERO  INTEGER NOT NULL,
    TITULO      VARCHAR(100),
    DURACAO     VARCHAR(10)
);
ALTER TABLE CLIENTES ADD PRIMARY KEY (COD_CLIENTE);
ALTER TABLE GENEROS ADD PRIMARY KEY (COD_GENERO);
ALTER TABLE LOCACAO ADD PRIMARY KEY (COD_LOCACAO);
ALTER TABLE USUARIOS ADD PRIMARY KEY (COD_USUARIO);
ALTER TABLE VIDEOS ADD PRIMARY KEY (COD_VIDEO);
ALTER TABLE LOCACAO ADD FOREIGN KEY (CLIENTE)
  REFERENCES CLIENTES (COD_CLIENTE)
  ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE LOCACAO ADD FOREIGN KEY (VIDEO)
  REFERENCES VIDEOS (COD_VIDEO)
  ON DELETE NO ACTION ON UPDATE NO ACTION;
ALTER TABLE VIDEOS ADD CONSTRAINT FK_VIDEOS
  FOREIGN KEY (COD_GENERO) REFERENCES
  GENEROS (COD_GENERO);
SET TERM ^ ;

CREATE TRIGGER CLIENTES_BI FOR CLIENTES
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_CLIENTE IS NULL) THEN
    NEW.COD_CLIENTE = GEN_ID(GEN_CLIENTES_ID,1);
END
^
CREATE TRIGGER GENEROS_BI FOR GENEROS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_GENERO IS NULL) THEN
    NEW.COD_GENERO = GEN_ID(GEN_GENEROS_ID,1);
END
^
CREATE TRIGGER LOCACAO_BI FOR LOCACAO
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_LOCACAO IS NULL) THEN
    NEW.COD_LOCACAO = GEN_ID(GEN_LOCACAO_ID,1);
END
^
CREATE TRIGGER USUARIOS_BI FOR USUARIOS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_USUARIO IS NULL) THEN
    NEW.COD_USUARIO = GEN_ID(GEN_USUARIOS_ID,1);
END
^
CREATE TRIGGER VIDEOS_BI FOR VIDEOS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD_VIDEO IS NULL) THEN
    NEW.COD_VIDEO = GEN_ID(GEN_VIDEOS_ID,1);
END
^
SET TERM ; ^
DESCRIBE FIELD STATUS TABLE LOCACAO
N = Normal
R = Reservado
D = Devolvido';
```

Digitado o *script*, basta executá-lo usando o botão *Run Script* ou pressione a tecla F9.

> **Nota:** Substitua *<Caminho>\Locadora.fdb* pelo diretório do banco de dados e seu nome, ex: *C:\Inetpub\wwwroot\Locadora.fdb*.

Após a execução do script teremos nosso banco de dados, tabelas, índices, chaves estrangeiras e tudo que é necessário para que nosso *BD* funcione perfeitamente.

Criando a aplicação

No BDS 2006 selecione o menu *File|New>ASP.NET Web Application – Delphi for .NET*. Na caixa de diálogo que aparece (**Figura 4**), defina o nome do projeto como "VideoLocadora", no item *Location* selecione o diretório destino onde os arquivos do projeto serão criados e no item *Server* defina o servidor de aplicação para a hospedagem do sistema.

Altere a página inicial, *WebForm1.aspx* para "principal.aspx", para isso clique com o botão direito do mouse no nome da página no *Project Manager* e selecione *Rename*. Antes de trabalharmos na página criada, iremos definir uma estrutura padrão para o cabeçalho do nosso sistema através de *User Control* na qual mais adiante faremos também o controle de usuários *logados*. Para adicionar o recurso é necessário criar uma página *.ascx* através do menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET User Control*. Dessa forma o Delphi fará a criação de uma nova página de edição do conteúdo. Altere no *Project Manager* o nome do *User Control* para "uccontrole.ascx" e na página adicione uma tabela com cerca de 700 *pixels* de largura contendo quatro linhas e uma coluna. Adicione às linhas da tabela o nome do sistema, uma imagem de identificação, um espaço para identificação do usuário *logado*, onde deverá adicionar também um componente *Label*("lblUsuario") e mais abaixo uma linha ("<hr>") para separação do cabeçalho ao conteúdo das páginas. Veja um exemplo de *layout* da **Figura 5.**

Retornando a página a *principal.aspx*, crie uma tabela de quatro linhas e uma coluna no centro da página com 700 *pixels* de largura. Arraste do *Project Manager* para dentro da primeira linha da tabela o *User Control uccontrole.ascx* que criamos. Já na segunda linha da tabela defina apenas um título para a localização do usuário, descreva como "Área Administrativa". Na linha três, adicione uma nova tabela para que possamos definir uma estrutura de menus para o sistema, crie uma tabela usando 100% da largura com quatro linhas e três colunas e adicione os seguintes componentes *Button* nas células da tabela: *Gêneros*("btnGeneros"), *Vídeos*("btnVideos"), *Clientes*("btnClientes"), *Usuários*("btnUsuarios"), *Locações*("btnLocacao"), *Lista de Reservas*("btnReservas") e *Sair*("btnSair"). Ainda na última linha da primeira tabela criada, adicione uma *tag* "*<hr>*" e os direitos de criação para sua aplicação. Clique duas vezes no *btnGeneros* e adicione o seguinte código, que ao ser clicado irá direcionar o usuário à devida página de manutenção dos gêneros de vídeos. O método *Redirect* sempre é usado quando necessitamos abrir uma outra página:

```
Response.Redirect('a_generos.aspx');
```

Como já codificamos o botão de manutenção dos gêneros de vídeos, veremos a seguir a criação da página e codificação para o cadastro de novos gêneros e manutenção de registros existentes. Crie uma nova página através do menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* e altere seu nome para "a_generos.aspx". Adicione ao corpo da página uma tabela centralizada com seis linhas, uma coluna e 700 *pixels* de largura. Arraste para a primeira linha o *User Control* da aplicação conforme realizado na criação da página principal. Na segunda linha defina o título da página para "Manutenção de Gêneros" e na linha seguinte adicione da *Tool Palette* um componente *DataGrid*("GridGeneros").

Faremos uso deste componente para visualizar os gêneros cadastrados no sistema. Mude a aparência do componente acessando a opção *Auto Format*, escolha um formato que lhe agrade e confirme. Em seguida clique novamente com o botão direito do mouse e escolha *Property Builder* ("Editor de Propriedades"). Nessa janela de configurações iremos

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como trabalhar com User Control no Delphi .NET.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=2220&hl=

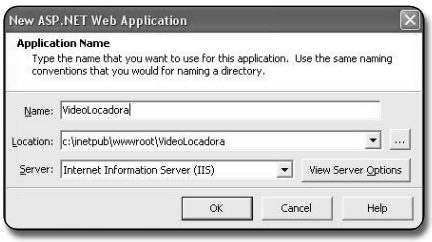


Figura 4. Criação da aplicação

Figura 5. Exemplo de cabeçalho

Figura 6. Exemplo de tela de Gêneros

Listagem 2. Código do evento Page_Load da página

```
interface
…
const
  strConexao = 'User=SYSDBA;Password=masterkey;
  Database=<Caminho>LOCADORA.FDB;';
…
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT COD_GENERO, GENERO FROM GENEROS';
  { Criaçã em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Genero');
  try
    GridGeneros.DataSource := Ds;
    GridGeneros.DataBind;
  finally
    Conn.Close;
  end;
end;
```

Listagem 3. Código do botão Incluir

```
procedure TWebForm1.btnSalvar_Click(sender:
  System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  prGenero : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'INSERT INTO GENEROS (GENERO) VALUES (?)';
  prGenero := FbParameter.Create;
  prGenero.ParameterName := 'GENERO';
  DataAdapter.InsertCommand.Parameters.Add(prGenero);
  DataAdapter.InsertCommand.Parameters[0].Value :=
    txtGenero.Text.ToUpper;

  if DataAdapter.InsertCommand.ExecuteNonQuery> 0 then
  begin
    { Atribuição dos atributos de seleção dos dados }
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText :=
      'SELECT COD_GENERO, GENERO FROM GENEROS';
    { Criaçã em memória do DataSet auxliar }
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Genero');
    try
      GridGeneros.DataSource := Ds;
      GridGeneros.DataBind;
      btnIncluir.Enabled := True;
      btnAlterar.Enabled := False;
      txtGenero.Text := '';
    finally
      Conn.Close;
    end;
  end;
end;
```

selecionar a categoria *Columns* e realizar as algumas alterações.

Desmarque, no topo da janela, o item *Create columns automatically at run time*. Em seguida clique no item *Bound Column* em *Available columns* e envie-o para *Selected columns*. Agora selecione-a e atribua o valor COD_GENERO na propriedade *Data Field*, onde receberemos o valor do campo na tabela *Generos* através do *result set* da *Select* que faremos. Desmarque também a opção *Visible* da coluna.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* ("Texto de cabeçalho"), digite "Gênero" e em *Data Field* o valor GENERO que é justamente o campo de nossa tabela *Generos* assim como no passo anterior.

Por fim inclua uma nova coluna de botões, chamada de *Button Column*. Expanda o *Button Column* e selecione a opção *Select*. Envie-o para a direita e modifique a propriedade *Header text* e *Text* para "Alterar" e clique em *Ok* para confirmar nossas alterações.

> **Nota**: Para alterar a largura das colunas, basta entrar novamente no *Property Builder* e selecionar o item *Format*. Nele podemos definir atributos para várias características do componente *Data Grid*. Note que temos a opção *Columns*. Expanda este item e clique na última coluna ("Alterar"). A direita digite 50 no campo *Width* ("largura") e confirme.

Na linha abaixo do *GridGeneros* digite o texto "Gênero:" e arraste um componente *TextBox*("txtGenero"). Na linha seguinte inclua três *Buttons*, sendo "btnIncluir", "btnAlterar" e "btnCancelar". O *btnAlterar* ficará com a propriedade *Enabled* desativada, ou seja, *False*. Por fim inclua um novo *Button*("btnVoltar") e em seu evento *OnClick* faça a chamada ao método *Redirect* conforme a seguir, para que possamos voltar a página principal (**Figura 6**):

```
Response.Redirect('a_principal.aspx');
```

Conectando o sistema ao banco de dados

Neste artigo iremos utilizar os *providers* do Firebird .NET Provider, porém faremos

todo o processo de conexão e atualização dos controles em tempo de execução. A primeira coisa que faremos é codificar o evento *Page_Load* da página, portanto pressione *F12* para visualizar a página de códigos e localize o evento citado. Digite o código da **Listagem 2** e vamos as explicações para entendermos melhor.

A primeira coisa que podemos ver na **Listagem 2** é que criamos uma constante na área *Interface* da página informando a *string* de conexão com o banco. Esse valor será usado mais adiante para que o componente *fbConnection* acesse o *BD*. Já no evento de carregamento da página, *Page_Load*, criamos quatro componentes em *run time* que são:

• *Conn*: Um *fbConnection* para conexão direta com o *BD*;

• *DataAdapter*: Componente para receber as instruções *SQL*;

• *Ds*: *DataSet* que recebe o *result set* das instruções *SQL* executadas;

• *Comand*: Componente *fbCommand* para execução de comandos;

Logo de início criamos todos os componentes e atribuímos a *string* de conexão com o banco ao *Conn*, componente de conexão. Em seguida atribuímos a conexão ao objeto interno do *DataAdapter* chamado *SelectCommand*. É ele que receberá a instrução *SQL* de seleção dos dados em sua propriedade *CommandText*.

Em seguida criamos um *DataSet*("Ds") e chamamos o método *Fill* do *DataAdapter* passando a ele a variável *Ds*("DataSet") e o nome da tabela ("Genero"). Feito isso modificamos a propriedade *DataSource* do *GridGeneros* informando qual seu *DataSet*, nesse caso a variável *Ds*. E para preencher o *grid* usamos o método *DataBind*.

O uso deste método melhora exponencialmente a performance da página, carregando-a muito mais rápido do que se usássemos os componentes diretamente na página. Vamos agora fazer a codificação dos botões *Alterar* e *Incluir*. Começaremos pelo botão *Incluir* clicando duas vezes nele e digitando o código da **Listagem 3**.

Perceba que utilizamos a mesma metodologia do evento *Page_Load*, ou seja, criamos os componentes todos em tempo de execução. A diferença aqui é que ao invés de usarmos o componente

Listagem 4. Código do botão Alterar

```
procedure TWebForm1.btnAlterar_Click(sender:
  System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  prCod_Genero : FbParameter;
  prGenero : FbParameter;
begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.ConnectionString := strConexao;
  Conn.Open;
  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText :=
    'UPDATE GENEROS SET GENERO = ?  WHERE (COD_GENERO = ?)';
  prCod_Genero := FbParameter.Create;
  prCod_Genero.ParameterName := 'COD_GENERO';
  prGenero := FbParameter.Create;
  prGenero.ParameterName := 'GENERO';
  DataAdapter.UpdateCommand.Parameters.
    Add(prCod_Genero);
  DataAdapter.UpdateCommand.Parameters.
    Add(prGenero);
  DataAdapter.UpdateCommand.Parameters[0].Value :=
    txtGenero.Text.ToUpper;
  DataAdapter.UpdateCommand.Parameters[1].Value :=
    GridGeneros.Items.Item[GridGeneros.SelectedItem.
    ItemIndex].Cells[0].Text;

  if DataAdapter.UpdateCommand.ExecuteNonQuery> 0 then
  begin
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText :=
      'SELECT COD_GENERO, GENERO FROM GENEROS';
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Genero');
    try
      GridGeneros.DataSource := Ds;
      GridGeneros.DataBind;
      btnIncluir.Enabled := True;
      btnAlterar.Enabled := False;
      txtGenero.Text := '';
    finally
      Conn.Close;
    end;
  end;
end;
```

Listagem 5. Código de Update

```
…
  DataAdapter.UpdateCommand.CommandText :=
    'UPDATE GENEROS SET GENERO = ?  WHERE (COD_GENERO = ?)';
  prCod_Genero := FbParameter.Create;
  prCod_Genero.ParameterName := 'COD_GENERO';
  prGenero := FbParameter.Create;
  prGenero.ParameterName := 'GENERO';
  DataAdapter.UpdateCommand.Parameters.Add(prCod_Genero);
  DataAdapter.UpdateCommand.Parameters.Add(prGenero);
  DataAdapter.UpdateCommand.Parameters[0].Value := txtGenero.Text.ToUpper;
  DataAdapter.UpdateCommand.Parameters[1].Value :=
    GridGeneros.Items.Item[GridGeneros.SelectedItem.ItemIndex].Cells[0].Text;
```

Listagem 6. Código do evento Page_Load de a_videos.aspx

```
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.ConnectionString := strConexao;
  Conn.Open;
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
   'SELECT VIDEOS.COD_VIDEO, VIDEOS.TITULO, ' +
   'GENEROS.GENERO,VIDEOS.DURACAO,GENEROS.COD_GENERO'+
   'FROM GENEROS' +
   'INNER JOIN VIDEOS ON (GENEROS.COD_GENERO = ' +
   'VIDEOS.COD_GENERO)';
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Genero');
  try
    GridVideos.DataSource := Ds;
    GridVideos.DataBind;
    { Atualiza o componente DropDownList }
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText :=
      'SELECT * FROM GENEROS';
    DataAdapter.Fill(Ds, 'Generos');
    with ddlGeneros do
    begin
      Items.Clear;
      DataTextField := 'GENERO';
      DataValueField := 'GENERO';
      DataSource := Ds;
      ddlGeneros.DataBind;
    end;
  finally
    Conn.Close;
  end;
end;
```

interno *SelectCommand*, estamos usando *InsertCommand*. Passamos para ele a instrução *SQL* de inclusão e criamos um parâmetro para que possa receber o nome do Gênero que será incluso. Veja as linhas a seguir:

```
  prGenero := FbParameter.Create;
  prGenero.ParameterName := 'GENERO';
  DataAdapter.InsertCommand.Parameters.
Add(prGenero);
  DataAdapter.InsertCommand.Parameters[0].
Value := txtGenero.Text.ToUpper;
```

Declaramos e criamos a variável *prGenero* que é do tipo *FbParameter*. Em seguida incluímos o parâmetro e o valor no *InsertCommand*. Tudo isso em função de nossa instrução de inclusão, veja:

```
INSERT INTO GENEROS (GENERO) VALUES (?)
```

Note o sinal de interrogação ao final de instrução. Para cada valor que precisamos receber em uma instrução *SQL* em ASP.NET incluímos o sinal. Veremos isso também no botão *Alterar*.

Voltando ao exemplo, chamamos o método *ExecuteNoQuery* e verificamos se o mesmo é maior que *0* ("zero") o que significa que a *Select* foi executada com sucesso. Para preenchermos novamente o *GridGenero* com os dados da tabela, repetimos o código do evento *Page_Load*, habilitamos e desabilitamos componentes conforme a necessidade.

Para realizar a alteração ("Update") de gêneros, faremos algo muito parecido com o código do botão *Incluir*. Veja a **Listagem 4** e implemente-a no evento *OnClick* do botão em questão. O código é muito semelhante, a diferença entre ambas, fiz questão de incluir em uma listagem separada, observe a **Listagem 5** que contém o trecho que comentarei.

Perceba que novamente mudamos o tipo de componente de comando usado no *DataAdpater*. Agora usamos *UpdateCommand* que executa comandos de *Update* no banco de dados. Como podemos perceber, o componente *Data Adapter* implementa quatro controles de acesso a dados. Sim, são quatro! Aqui estamos usando apenas *SelectCommand*, *InsertCommand* e *UpdateCommand*, porém ainda temos a opção de usar o *DeleteCommand*, para comandos de exclusão.

Na explicação do botão *Incluir*, vimos como usar parâmetros no ASP.NET. Se observarmos com mais calma o código da **Listagem 4**, veremos que estamos usando dois parâmetros: *COD_GENERO* e *GENERO*. Veja:

```
UPDATE GENEROS SET GENERO = ?  WHERE (COD_
GENERO = ?)
```

Para usar esses parâmetros precisamos criar duas variáveis e inicilizá-las, assim como podemos ver no trecho adiante.

```
  prCod_Genero := FbParameter.Create;
  prCod_Genero.ParameterName := 'COD_GENERO';
  prGenero := FbParameter.Create;
  prGenero.ParameterName := 'GENERO';
```

Por fim enviamos essas variáveis ao *UpdateCommand* e executamos. O restante do código faz o *reload* dos dados no *DataGrid*.

Nota: Para que possamos criar os componentes da paleta *Firebird Data Provider* em tempo de execução, é necessário declarar o *namespace FirebirdSql.Data.Firebird* ao *Uses* da página.

Criando a tela de manutenção de vídeos

Finalizada a construção, codificação e testes da página de *Gêneros* veremos agora a criação da página de manutenção de vídeos que envolve a utilização de valores de outra tabela do banco de dados, a tabela *Videos*.

O desenvolvimento da página também se dá de forma bastante simples e utilizaremos mecanismos semelhantes ao processo anterior. Após a criação desta página, baseado nestes dois exemplos, faça você mesmo a criação das páginas de clientes e usuários do sistema.

Crie uma nova página no menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* e altere seu nome para "a_generos.aspx". Insira no corpo da página uma tabela de seis linhas e uma coluna com 700 *pixels* de largura onde iremos adicionar os componentes para o controle da nossa página. Na primeira linha da tabela arraste *uccontrole.ascx*. Escreva

na próxima linha o texto "Manutenção de Vídeos" para identificar a página e na linha seguinte adicione um componente *DataGrid* ("gridVideos") e acesse no *Object Inspector* o item *Property Builder* posicionando-se na guia *Columns* onde deverão ser observadas algumas configurações.

Desmarque, no topo da janela, o item *Create columns automatically at run time*. Em seguida clique no item *Bound Column* em *Available columns* e envie-o *Selected columns*. Agora selecione-a e atribua o valor TITULO nas propriedades *Data Field* e *Header text*, onde receberemos o valor do campo na tabela *Videos* através do *result set* da *Select* que faremos. Lembre-se que a propriedade *Header text* é na verdade o texto que aparecerá para o usuário final de sua aplicação.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* digite "Gênero" e em *Data Field* o valor GENERO. Repita os passos anteriores para o campo DURACAO, COD_VIDEO e COD_GENERO, porém esses dois últimos deverão ter a propriedade *Visible* desativada.

Por fim faremos a inclusão de uma coluna de botões, chamada de *Button Column*. Expanda o *Button Column* e selecione a opção *Select*. Envie-o para a direita e modifique a propriedade *Header text* e *Text* para "Alterar" e clique em *Ok* para confirmar todas as alterações.

Após isso, insira uma nova tabela com três linhas e duas colunas com 100% de largura, na linha logo abaixo. Digite o texto "Vídeo: " na primeira célula e na coluna ao lado arraste um componente *TextBox*("txtVideo"). Agora digite o texto "Gênero: " na célula seguinte e arraste um componente *DropDownList* ("ddlGeneros"). Por fim, na última linha, utilize o texto "Duração: " e adicione outro controle *TextBox*("txtDuracao").

Nas próximas linhas configure como na página de manutenção de gêneros. No código do botão *btnVoltar*, evento *OnClick*, digite o código a seguir:

```
Response.Redirect('a_principal.aspx');
```

Usaremos os mesmos métodos de conexão e listagem de dados do nosso *BD* usados na tela de *Gêneros*, ou seja, criaremos todos os objetos em *run time*.

Listagem 7. Código do botão Alterar

```
procedure TWebForm1.btnAlterar_Click(sender:
  System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  prCod_Genero : FbParameter;
  prTitulo : FbParameter;
  prDuracao : FbParameter;
  prCod_Video : FbParameter;
begin
  Item := ddlGeneros.SelectedValue.ToString;
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.ConnectionString := strConexao;
  Conn.Open;
  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText :=
    'UPDATE VIDEOS SET COD_GENERO =?, TITULO =?,'+
    'DURACAO =? WHERE (COD_VIDEO =?)';

  prCod_Genero := FbParameter.Create;
  prCod_Genero.FbDbType := FbDbType(9);
  prCod_Genero.ParameterName := 'COD_GENERO';
  DataAdapter.UpdateCommand.Parameters.Add(
    prCod_Genero);
  DataAdapter.UpdateCommand.Parameters[0].Value :=
    ddlGeneros.SelectedValue.ToString;

  prTitulo := FbParameter.Create;
  prTitulo.ParameterName := 'TITULO';
  DataAdapter.UpdateCommand.Parameters.Add(prTitulo);
  DataAdapter.UpdateCommand.Parameters[1].Value :=
    txtVideo.Text.ToUpper;

  prDuracao := FbParameter.Create;
  prDuracao.ParameterName := 'DURACAO';
  DataAdapter.UpdateCommand.Parameters.Add(prDuracao);
  DataAdapter.UpdateCommand.Parameters[2].Value :=
    txtDuracao.Text;

  prCod_Video := FbParameter.Create;
  prCod_Video.FbDbType := FbDbType(9);
  prCod_Video.ParameterName := 'COD_VIDEO';
  DataAdapter.UpdateCommand.Parameters.Add(
    prCod_Video);
  DataAdapter.UpdateCommand.Parameters[3].Value :=
    gridVideos.Items.Item[gridVideos.SelectedItem
    .ItemIndex].Cells[4].Text.ToString;

  if DataAdapter.UpdateCommand.ExecuteNonQuery> 0 then
  begin
    { Atribuição dos atributos de seleção dos dados }
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText :=
      'SELECT VIDEOS.COD_VIDEO, VIDEOS.TITULO,'+
      'GENEROS.GENERO, VIDEOS.DURACAO, '+
      'GENEROS.COD_GENERO FROM GENEROS' +
      '   INNER JOIN VIDEOS ON ' +
      ' (GENEROS.COD_GENERO = VIDEOS.COD_GENERO)';
    { Criação em memória do DataSet auxliar }
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Genero');
    try
      GridVideos.DataSource := Ds;
      GridVideos.DataBind;
      btnIncluir.Enabled := True;
      btnAlterar.Enabled := False;
      txtVideo.Text := '';
      txtDuracao.Text := '';
    finally
      Conn.Close;
    end;
  end;
end;
```

Listagem 8. Código do botão Incluir da tela de Vídeos

```
procedure TWebForm1.btnIncluir_Click(sender:
  System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  prCod_Genero : FbParameter;
  prTitulo : FbParameter;
  prDuracao : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'INSERT INTO VIDEOS (COD_GENERO, TITULO, DURACAO)
    VALUES (?, ?, ?)';

  prCod_Genero := FbParameter.Create;
  prCod_Genero.FbDbType := FbDbType(9);
  prCod_Genero.ParameterName := 'COD_GENERO';
  DataAdapter.InsertCommand.Parameters.Add(
    prCod_Genero);
  DataAdapter.InsertCommand.Parameters[0].Value :=
    ddlGeneros.SelectedValue.ToString;

  prTitulo := FbParameter.Create;
  prTitulo.ParameterName := 'TITULO';
  DataAdapter.InsertCommand.Parameters.Add(
    prTitulo);
  DataAdapter.InsertCommand.Parameters[1].Value :=
    txtVideo.Text.ToUpper;

  prDuracao := FbParameter.Create;
  prDuracao.ParameterName := 'DURACAO';
  DataAdapter.InsertCommand.Parameters.Add(prDuracao);
  DataAdapter.InsertCommand.Parameters[2].Value :=
    txtDuracao.Text;

  if DataAdapter.InsertCommand.ExecuteNonQuery> 0 then
  begin
    { Atribuição dos atributos de seleção dos dados }
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText :=
      'SELECT VIDEOS.COD_VIDEO, VIDEOS.TITULO, '+
      'GENEROS.GENERO,' +
      'VIDEOS.DURACAO, GENEROS.COD_GENERO FROM '+
      'GENEROS' +
      '   INNER JOIN VIDEOS ON (GENEROS.COD_GENERO'+
      ' = VIDEOS.COD_GENERO)';
    { Criação em memória do DataSet auxliar }
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Genero');
    try
      GridVideos.DataSource := Ds;
      GridVideos.DataBind;
      btnIncluir.Enabled := True;
      btnAlterar.Enabled := False;
      txtVideo.Text := '';
      txtDuracao.Text := '';
    finally
      Conn.Close;
    end;
  end;
end;
```

Digite o código da **Listagem 6** ao evento *Page_Load* da página. As únicas diferenças aqui são o nome do componente *DataGrid*, que será modificado para *GriVideos* e instrução *SQL* que estamos usando para trazer os dados. Precisamos listar todos os vídeos presentes na locadora e seus respectivos gêneros. Para isso usamos um *JOIN* entre as tabelas *Vídeos* e *Generos*. Para facilitar nossa a programação, volte na página *a_generos.aspx* e copie o código completo do evento *Page_Load*. Após isso cole-o no *Page_Load* de nossa página. Altere a instrução *SQL* como mostrado a seguir:

```
DataAdapter.SelectCommand.CommandText :=
  'SELECT VIDEOS.COD_VIDEO,
     VIDEOS.TITULO, GENEROS.GENERO,' +
  'VIDEOS.DURACAO, GENEROS.COD_GENERO FROM
  GENEROS' +
  '   INNER JOIN VIDEOS ON (
  GENEROS.COD_GENERO = VIDEOS.COD_GENERO)';
```

Em seguida modifique, no código-fonte, o nome do componente *DataGrid* de *GridGeneros* para *GridVideos*. Só há mais um detalhe que devemos nos lembrar. Inserimos um *DropDownList*("ddlGeneros") onde listaremos todos os gêneros da tabela para que possamos usá-lo na tela de manutenção de vídeos. Repare que entre o *try..finally* inclui um trecho de código que faz uma nova *Select* no banco, traz todos os gêneros e em seguida preenche a propriedade *Items* do *ddlGenereos*. O código completo do evento você pode conferir da **Listagem 6** como dito anteriormente.

A alteração de vídeos é simples, embora com uma quantidade de código extensa. A **Listagem 7** mostra como fazer a alteração do item selecionado. Basicamente o código é bem semelhante ao de inclusão. Assim como fizemos no botão de alteração de gêneros, também devemos fazer em vídeos, ou seja, criamos todos os componentes em *run time*, modificamos a instrução *SQL* para fazer o *Update* dos campos e trocamos o componente de comando no *DataAdapter* de *InsertCommand* para *UpdateCommand*. Também somos obrigados a criar um novo parâmetro, com o valor do campo *COD_VIDEO* para que nossa instrução *SQL* saiba em qual registro do banco as alterações serão aplicadas. Veja nossa instrução a seguir:

```
UPDATE VIDEOS SET COD_GENERO =?, TITULO =?,
DURACAO =? WHERE (COD_VIDEO =?)
```

Por fim é necessário fazer com que os dados do vídeo selecionado sejam enviados aos controles de tela. Para isso digite o código a seguir no evento *ItemCommand* do *GridVideos*. Perceba que apenas atribuímos os valores das células selecionadas aos controles de tela. Depois habilitamos e desabilitamos os componentes necessários.

```
txtVideo.Text    := e.Item.Cells.Item[0].Text;
txtDuracao.Text := e.Item.Cells.Item[2].Text;
ddlGeneros.SelectedValue := e.Item.Cells.Item[5].Text;
btnAlterar.Enabled := True;
btnIncluir.Enabled  := False;
```

Faremos agora a última parte de desenvolvimento da página de manutenção de vídeos, a codificação do botão *Incluir*. O esquema é o mesmo do botão *Alterar*, por isso podemos copiar e colar o código do evento *OnClick* dele e colocar no botão *Incluir*. Depois faremos algumas modificações para adaptar o código às nossas necessidades de inclusão. Devemos trocar o componente *UpdateCommand* por *InsertCommand* e incluir a instrução *SQL* para inserção (**Listagem 8).**

Realizada toda a codificação do processo de inclusão e alteração da página de manutenção de vídeos, volte para a página *principal.aspx* e adicione ao evento *OnClick* do botão *btnVideos* o seguinte código:

```
Response.Redirect('a_videos.aspx');
```

Execute sua aplicação e faça a simulação de cadastro de novos vídeos e também da alteração do cadastro de vídeos já existentes. A partir da criação da página de gêneros e vídeos, baseando-se nestes exemplos crie uma página de cadastro e manutenção para usuários que serão os administradores do sistema e também uma página para cadastros e manutenções de registros de clientes, levando em consideração que cada cliente poderá ter um usuário e senha para fazer suas próprias reservas *on-line*. Veja na **Figura 7** o sistema em execução.

Figura 7. Sistema em execução

Conclusão

Vimos nesta primeira parte do artigo a criação e configuração da conexão com a base de dados. Também a criação das páginas de cadastro para gêneros e vídeos, utilizando valores de outras tabelas.

Faça sua própria implementação de chaves de criptografia se preferir e construa funções e procedimentos para validação de registros cadastros e não permitindo redundâncias. Acompanhe no próximo artigo a criação da página de reservas locais, controle de usuários e clientes *logados*, reservas *on-line* e posterior efetivação da retirada do vídeo reservado. Grande abraço e até o próximo artigo. ●

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Conhecendo e personalizando o componente StatusBar

Saiba tudo sobre o componente StatusBar e personalize-o como desejar

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Na seção Easy Delphi passada falei sobre alguns dos componentes, funções e procedimentos para exibição de mensagens ao usuário final. Vimos como utilizar *ShowMessage*, *MessageDlg* e *MessageBox*. Passamos também por componentes de diálogo, tais como: *OpenDialog*, *SaveDialog* etc. Nessa seção quero falar sobre outro componente bastante interessante e que acredito não ser tão abordado no dia-a-dia quanto os demais já comentados. Trata-se do *StatusBar*. Veremos nesse artigo algumas das principais características desse controle que sem dúvida nenhuma pode se tornar um poderoso aliado de sua aplicação.

Em nossa aplicação exemplo faremos uso do *StatusBar* e aprenderemos a:

- Mostrar *Hint's* de toda a aplicação, incluindo botões e menus, diretamente na barra de Status;
- Desenhar imagens na barra;
- Incluir eventos para interagir como o usuário;
- Incluir barra de progresso.

Todos os recursos descritos anteriormente podem ser facilmente implementados a aplicação sem auxílio de nenhuma biblioteca de terceiro. Tudo nativo do Delphi. Mão na massa e vamos aos exemplos práticos.

Entendendo a barra de Status (StatusBar)

Para começar vejamos primeiramente o que exatamente é a barra de status e quais suas utilidades. A barra de status, ou *StatusBar*, está presente na VCL do Delphi desde as primeiras versões da ferramenta e é utilizada como uma barra auxiliar em diversos aplicativos, principalmente para ser mostrar mensagens ou o estado de alguma situação. Podemos citar como exemplos típicos os aplicativos Windows Explorer e Word da

Microsoft, que se utiliza de barras de status para mostrar dados do disco rígido e do documento atual, respectivamente. (**Figuras 1** e **2**)

Perceba que o rodapé do Word e do Windows Explorer mostra informações importantes para o usuário, tais como número da página atual e o número total de páginas ("Word") e o espaço em disco disponível ("Windows Explorer"). Além disso, é perfeitamente possível incluir imagens, barras de progresso e até interagir com o usuário por meio de eventos, ou seja, o permitir ações ao clicar na barra.

Para iniciar o uso do *StatusBar* vejamos como fazer como que a aplicação inteira mostre o *Hint* de ajuda diretamente na barra. Criaremos uma aplicação simples e adicionaremos recursos mais avançados a *StatusBar*.

Nota do DevMan

Todo componente visual possui uma propriedade Hint usada para mostrar mensagens ao usuário da aplicação. Por padrão, os hint's são exibidos em forma de pequenas caixas amarelas quando passamos o mouse sobre um objeto.

Criando a aplicação exemplo

Crie uma nova aplicação no Delphi 7.0 ou na versão que preferir e salve seu formulário principal como "uPrincipal.pas" usando o menu *File|Save as*. Modifique a propriedade *Name* do formulário para "frmPrincipal" e seu *Caption* para "Trabalhando com Status Bar". Em seguida salve a aplicação como "StatusBar.dpr" usando o menu *File|Save project as*. Inclua um componente *MainMenu*("mnuPrincipal") da paleta *Standard*. Modifique o *mnuPrincipal* incluindo itens menu conforme a estrutura da **Figura 3**.

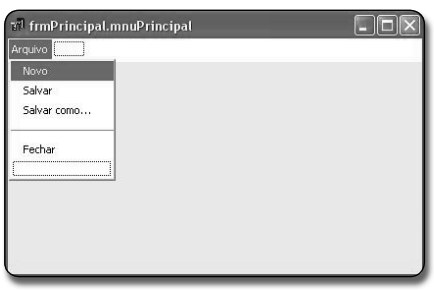


Figura 3. Estrutura do menu de exemplo

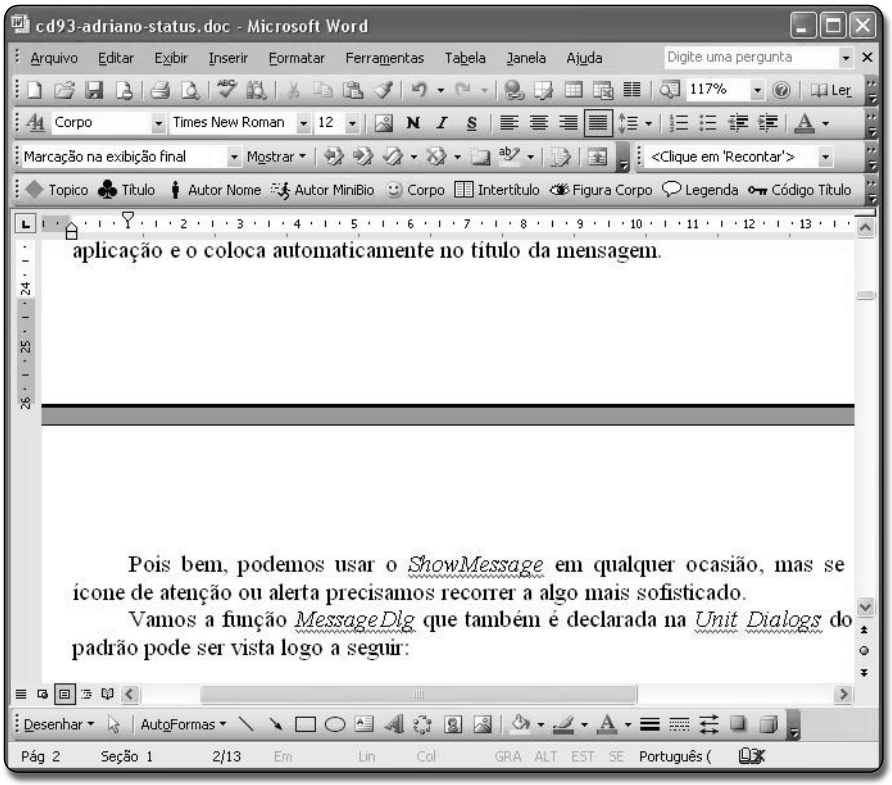


Figura 1. Barra de status no Microsoft Word

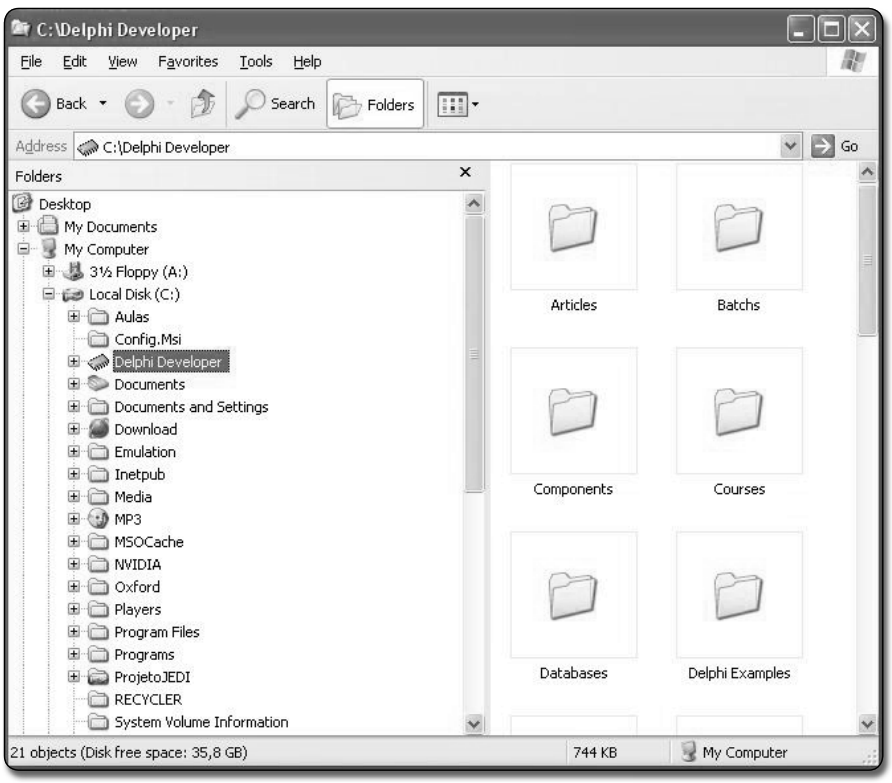


Figura 2. Barra de status no Microsoft Windows Explorer

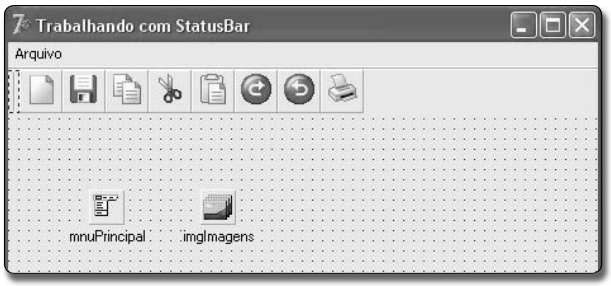


Figura 4. Exemplo de tela principal com botões adicionados

Figura 5. Hint do botão salvar na aplicação em execução

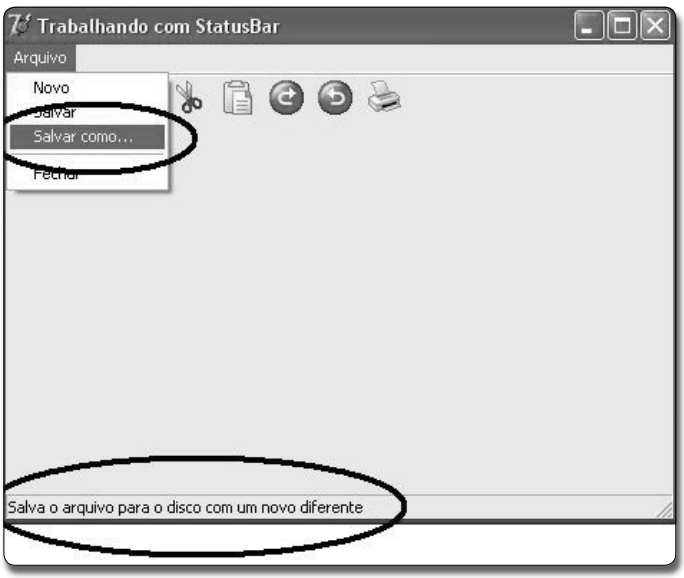


Figura 6. Hint de menu sendo mostrado na StatusBar

Criados os itens de menu selecione cada um deles e inclua as frases a seguir na propriedade *Hint* de dada item seguindo a ordem de *Arquivo* para *Fechar*:

• *Arquivo*: Menu com todas as operações em arquivos;
• *Novo*: Cria um novo arquivo no disco;
• *Salvar*: Salva o arquivo para o disco;
• *Salvar como*: Salva o arquivo para o disco com um novo diferente;
• *Fechar*: Fechar este aplicativo.

Adicione ao formulário um componente *ToolBar*("tobBotoes") e um ImageList("imgImagens") da paleta *Win32*. Altere as propriedades *Heigth* e *Width* do *tobBotoes* para 24 em ambas. Modifique também a propriedade *Images* apontando para *imgImagens*. Por último altere a propriedade *Flat* para *True*, o que faz o botão ficar achatado.

Inclua algumas imagens no tamanho, 24x24 se disponível, ao componente *imgImagens*. Em seguida clique com o botão direito do mouse no *tobBotoes* e selecione o item *New Button* para adicionar novos botões à barra. Caso seja necessário modifique a propriedade *ImageIndex* de cada botão indicando qual o número da imagem o botão fará referência no *imgImagens*. Para uma melhor aparência dos botões, aumente um deles para que tenha o tamanho igual a 34x34. Veja um exemplo na **Figura 4**.

Clique no formulário e marque como *True* a propriedade *ShowHint*, isso fará com que todos os *Hint's* sejam exibidos. Para finalizar, clique em cada botão e escreva mensagens de ajuda na propriedade *Hint* de cada um.

Execute a aplicação e passe o mouse pelos botões e itens de menu. Perceba que pequenas mensagens vão aparecendo (**Figura 5**), porém nos menus isso não acontece. É aí que entra a *StatusBar*.

Os *Hint's* na aplicação são gerenciados pelo o objeto *Application*, da classe *TApplication*. Esse objeto gerencia toda a aplicação e possui alguns eventos interessantes, como por exemplo *OnHint*. Podemos criar uma pequena função que atualiza a *StatusBar* com a mensagem do componente em foco. Então desviamos o evento *OnHint* para essa função.

Falando dessa forma parece ser super difícil, um bicho de sete cabeças, mas não é. Vejamos como fazer isso. A primeira coisa a se fazer é incluir um componente *StatusBar*("stbMensagens") da paleta Win32 no formulário caso ainda não o tenha feito.

Por padrão a barra de status fica "colada" na parte inferior do formulário, como é comumente usado, porém é perfeitamente possível mudar sua direção através da propriedade *Align*, que possui algumas variantes, entre elas *alTop*("ao topo"), *alLeft*("a esquerda") e *alRight*("a direita").

Agora faremos uma pequena modificação no código-fonte da nossa aplicação.

Codificando o exemplo

Pressione *F12* para visualizar o *Code Editor* e localize a seção *private* no código-fonte. Declare o procedimento a seguir e pressione *Ctrl + Shift + C* para seja criado o cabeçalho da função.

```
procedure MostrarHint(Sender: TObject);
```

No corpo do procedimento apenas digite:

```
stbMensagens.SimpleText := Application.Hint;
```

O que estamos fazendo é muito simples. Apenas pegamos o valor da propriedade *Hint* do objeto *Application* e atualizamos a propriedade *SimpleText* do *StatusBar*. Feito isso o sistema pegará automaticamente o

Figura 7. Diálogo de painéis da StatusBar

Listagem 1. Evento OnTimer do Timer

```
procedure TfrmPrincipal.Timer1Timer(Sender: TObject);
begin
  if Odd(GetKeyState(VK_CAPITAL)) then
    stbMensagens.Panels[1].Text := 'CAP'
  else
    stbMensagens.Panels[1].Text := '';
  if Odd(GetKeyState(VK_NUMLOCK)) then
    stbMensagens.Panels[2].Text := 'NUM'
  else
    stbMensagens.Panels[2].Text := '';
end;
```

Listagem 2. Código do evento OnDrawPanel

```
procedure TfrmPrincipal.stbMensagensDrawPanel(
  StatusBar: TStatusBar; Panel: TStatusPanel; const Rect: TRect);
begin
  with stbMensagens.Canvas do
  begin
    FillRect(Rect);
    Font.Name := 'Verdana';
    Font.Color := clnavy;
    Font.Style := [fsBold, fsItalic];
    imgImagens.Draw(stbMensagens.Canvas,Rect.Left + 5, Rect.Top + 1, 8);
    if Panel.index = 3 then
      TextOut(Rect.Left + 35, Rect.Top + 5, 'Devmedia Editora');
  end;
end;
```

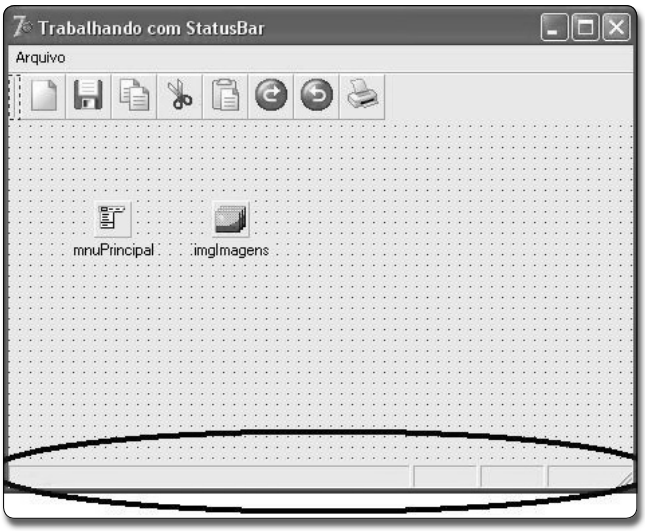


Figura 8. Barra de status com painéis

Hint do objeto selecionado e copiará para nosso *StatusBar*. Mas para que isso funcione corretamente, teremos que interceptar o evento *OnHint* da aplicação e desviá-lo para nosso procedimento *MostrarHint*.

Volte no formulário, selecione o evento *OnCreate* e clique duas vezes. No *OnCreate* inclua a linha a seguir:

```
Application.OnHint := MostrarHint;
```

Note, estamos "dizendo" ao objeto *Application* que quem fará a gerência dos *Hint's* do sistema agora é a função *Mostrar Hint*. Execute novamente a aplicação e passe o mouse pelos botões e menus como feito anteriormente. Observe que todas as mensagens estão sendo mostradas no rodapé do nosso exemplo, incluindo menus (**Figura 6**).

Trabalhando com mais de uma mensagem

Muito bem, mas a *StatusBar* é mais poderosa do que isso, e é possível mostrarmos mais que uma mensagem ao mesmo tempo, assim como os aplicativos já citados: Windows Explorer e Word.

Vejamos agora como trabalhar com painéis que é a parte mais interessante do artigo. O controle *StatusBar* pode ser dividido em painéis, chamados de *Panels*. Se clicar duas vezes no componente verá que uma pequena caixa de diálogo aparece (**Figura 7**).

Para adicionar painéis, basta clicar no botão ativo na barra de tarefas da caixa de diálogo. Cada clique no botão adiciona um painel, e cada painel possui suas propriedades. As principais e mais importantes são:

- *Alignment*: Alinha o texto do painel;
- *Style*: Estilo do painel. Podemos escolhe somente *Texto*("psText") ou *Modo Desenho*("psOwnerDraw");
- *Text*: Armazena um texto, seja do *Hint* ou de qualquer outro lugar;
- *Width*: Largura do painel.

Agora que conhecemos as principais propriedades dos painéis, insira quatro painéis no *StatusBar* sendo o primeiro com largura igual a 350 e os demais igual a 30. A aparência de nossa aplicação ficará como na **Figura 8**.

Certamente se executarmos o exemplo agora perceberemos que os painéis somem ao iniciar. Isso acontece porque estamos atualizando a propriedade *SimpleText* do *stbMensagens* e estamos trabalhando com painéis. O certo é incluirmos a mensagem diretamente na propriedade *Text* do *painel*. Para isso teremos que fazer uma pequena alteração em nosso procedimento *MostrarHint*. Pressione F12 para visualizar o código e localize o procedimento. Altere a propriedade *SimpleText* para *Panels[0].Text* conforme adiante:

```
stbMensagens.Panels[0].Text := Application.Hint;
```

Os painéis no componente são representados pela propriedade *Panels* onde indicamos o seu *Index*("índice") iniciando em zero, ou seja, o primeiro painel é *0*, o segundo é *1* e assim sucessivamente. O painel por sua vez possui algumas propriedades como vimos anteriormente. Uma delas certamente é o *Text*, então atualizamos a sua propriedade usando o atributo *Hint* do objeto *Application*.

Vamos avançar um pouquinho mais

Figura 9. Imagem e painéis coloridos

Listagem 3. Modificações no evento OnDrawPanel

```
procedure TfrmPrincipal.stbMensagensDrawPanel(
  StatusBar: TStatusBar; Panel: TStatusPanel; const Rect: TRect);
begin
  with stbMensagens.Canvas do
  begin
    FillRect(Rect);
    Font.Name := 'Verdana';
    Font.Color := clnavy;
    Font.Style := [];
    case Panel.Index of
      1:
        begin
          if (Panel.Text = 'CAP') then
            Brush.Color := clRed
          else
            Brush.Color := clBtnFace;
          TextRect(Rect, Rect.Left + 10, Rect.Top + 5, Panel.Text);
        end;
      2:
        begin
          if (Panel.Text = 'NUM') then
            Brush.Color := clYellow
          else
            Brush.Color := clBtnFace;
        end;
    end;
    FillRect(Rect);
    if Panel.Index in [1, 2] then
      TextRect(Rect, Rect.Left + 10, Rect.Top + 5, Panel.Text)
    else
    begin
      imgImagens.Draw(stbMensagens.Canvas, Rect.Left + 5, Rect.Top + 1, 8);
      TextOut(Rect.Left + 35, Rect.Top + 5, 'Devmedia Editora');
    end;
  end;
end;
```

Listagem 4. Código do OnCreate modificado

```
procedure TfrmPrincipal.FormCreate(Sender: TObject);
var
  ProgressBarStyle: integer;
begin
  Application.OnHint := MostrarHint;
  ProgressBar1 := TProgressBar.Create(frmPrincipal);
  ProgressBar1.Parent := stbMensagens;
  ProgressBarStyle:=GetWindowLong(ProgressBar1.Handle, GWL_EXSTYLE);
  ProgressBarStyle := ProgressBarStyle - WS_EX_STATICEDGE;
  SetWindowLong(ProgressBar1.Handle, GWL_EXSTYLE, ProgressBarStyle);
end;
```

e pedir ajuda as API's do Windows pra mostrar se as teclas *Caps Lock* e *Num Lock* estão pressionadas. Inclua um componente *Timer* da paleta *System* no formulário e clique duas vezes nele. Em seu evento *OnTimer* codifique-o conforme a **Listagem 1**.

A função *Odd* nos retorna se um número é ímpar. O que estamos fazendo no evento é capturar os valores das constantes *VK_CAPITAL*("Caps Lock") e *VK_NUMLOCK*("Num Lock") checar seu estado usando a API *GetKeyState* que verifica o estado de uma tecla. Caso o valor seja ímpar, significa que a tecla foi pressionada então mostramos *CAP* para *Caps Lock* e *NUM* para *Num Lock*. Para *Caps Lock* estamos usando o painel *1* e para *Num Lock* o painel *2*. Execute a aplicação e experimente pressionar várias vezes *Num Lock* e *Caps Lock*.

Desenhando imagens

Para incrementar ainda mais nossa *StatusBar* vamos adicionar o logotipo da empresa no último painel da barra, porém precisaremos aumentá-la um pouco em sua altura. Deixa-a com *30 px* de altura e modifique a propriedade *Style* do último painel para *psOwnerDraw*. Inclua uma nova imagem de *24x24 px* no *ImageList* e vamos codificar um dos eventos da barra.

Em meu exemplo inclui oito imagens que vão de *0* a *7*. Para esse exemplo inclui uma nova imagem, portanto o índice da imagem no *ImageList* passa a ser *8*. Já entenderemos o porque saber qual o índice na nova imagem. Selecione o *StatusBar* e clique duas vezes no evento *OnDrawPanel*. Em seguida digite o código da **Listagem 2**.

Basicamente estamos desenhando no *Canvas* da *StatusBar*. O *Canvas*, grosseiramente falando, é uma camada dos controles no Windows. É como se estivéssemos desenhando sobre o própria imagem no controle. *TRect* é uma classe que cria um retângulo, no caso da *StatusBar* é criado automaticamente um retângulo invisível para cada painel. Por isso usamos o método *FillRect* para preenchê-lo. Em seguida modificamos as propriedades de fonte e "mandamos" desenhar a imagem *8* do nosso *ImageList*

que exatamente a última imagem que adicionamos. Ele está no índice *8*, o último. Por mim verificamos se é o painel *3* e então desenhamos a mensagem "Devmedia Editora".

E já que estamos falando em desenhar, que tal colorimos os painéis onde aparecem o status das teclas *Num Lock* e *Caps Lock*? Pois bem, vamos lá. Precisaremos mudar radicalmente o evento *OnDrawPanel*, portanto volte ao código do evento e refaça-o inteiro como mostrado na **Listagem 3**.

A mudança realmente foi grande, porém nada complexo. Criamos um *case.. of* onde testamos qual painel está sendo pintado no momento. Se for o painel *1*, que mostra o estado do *Caps Lock*, verificamos se o mesmo está pressionado então pintamos de vermelho, caso contrário voltamos a pintura para a cor padrão, *clBtnFace*. O mesmo fazemos com o painel que mostra o estado da tecla *Num Lock*. Por fim verificamos se é o painel 1 ou 2 para então pintar o texto do próprio painel, ou seja, *CAP* ou *NUM*. Caso contrário pintamos o texto e a imagem da nossa empresa (**Figura 9**).

Incluindo um ProgressBar

Vamos complicar um pouquinho mais, e ter uma barra de status mais avançada. Incluiremos uma *ProgressBar*("barra de progresso") em um painel de nossa *StatusBar*. Inclua um novo painel a *StatusBar* e aumente a largura do formulário. Aumente também a largura do penúltimo painel, onde encontra-se nosso logotipo, para o tamanho 200 px. Vá até a página de código e encontre a seção *public*. Declare a variável *ProgressBar1* do tipo *TProgressBar*, como segue:

```
public
  ProgressBar1 : TProgressBar;
end;
```

Agora vá até o evento *OnCreate* do formulário e modifique-o conforme a **Listagem 4**. Perceba que estamos criando uma barra de progresso em tempo de execução diretamente na barra de status. Isso é feito informando o *Parent* da *ProgressBar* que nesse caso ficou sendo o *StatusBar*("stbMensagens").

Em seguida recorremos as API's do Windows para alterar o formato da

Listagem 5. Inclusão de código no evento OnDrawPanel

```
…
  if Panel = StatusBar.Panels[4] then
  with ProgressBar1 do
  begin
    Top := Rect.Top;
    Left := Rect.Left;
    Width := Rect.Right - Rect.Left - 15;
    Height := Rect.Bottom - Rect.Top;
  end;
...
```

Listagem 6. Evento OnClick do Botão de teste

```
procedure TfrmPrincipal.Button1Click(Sender:
  TObject);
var
  I : integer;
begin
  ProgressBar1.Position := 0;
  ProgressBar1.Max := 100;
  for i := 0 to 100 do
  begin
    ProgressBar1.Position := i;
    Sleep(25);
  end;
end;
```

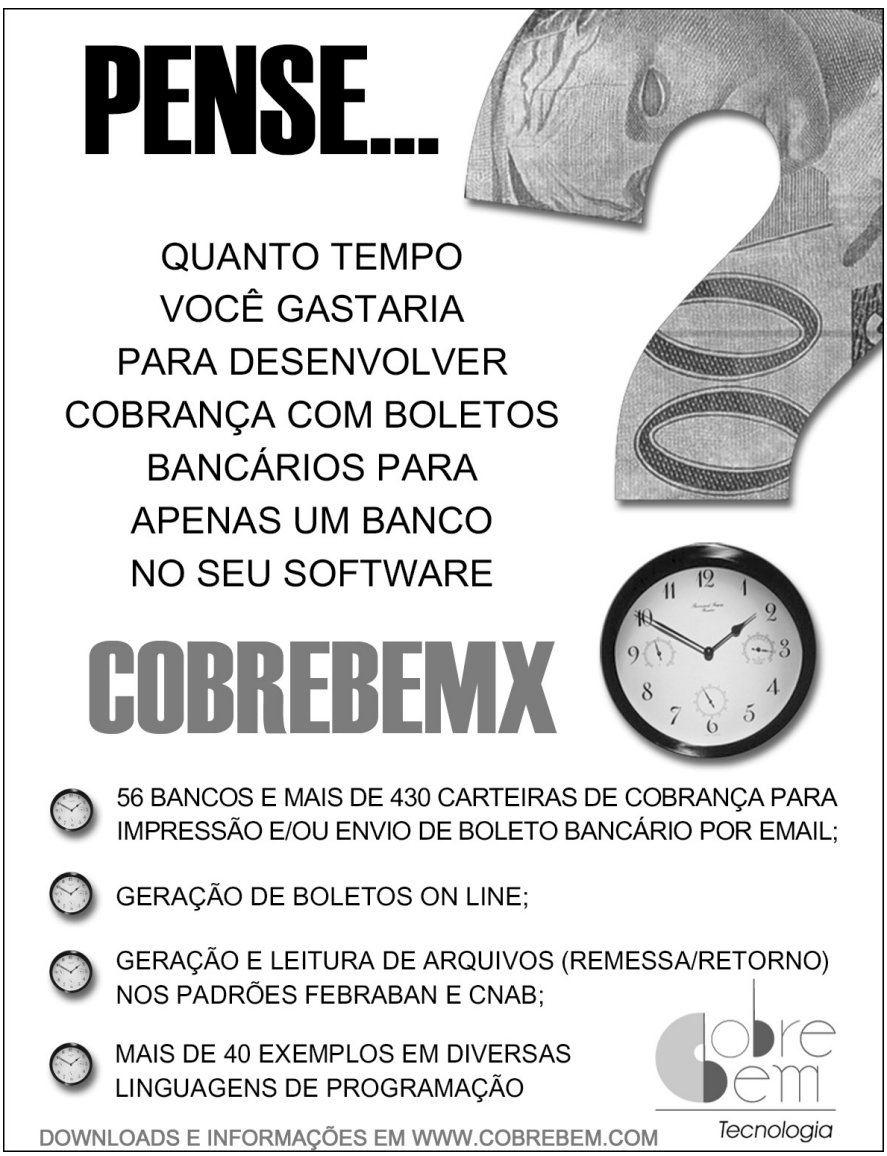



Figura 10. Barra de progresso na StatusBar

Figura 11. Capturando painel clicado

Listagem 7. Código OnClick da StatusBar

```
procedure TfrmPrincipal.stbMensagensClick(Sender: TObject);
var
  PosicaoMouse : TPoint;
  X : Integer;
  J : Integer;
  Painel : integer;

begin
  PosicaoMouse := Mouse.CursorPos;
  PosicaoMouse := stbMensagens.ScreenToClient(
    PosicaoMouse);
  Painel := -1;
  X := 0;

  for J := 0 to stbMensagens.Panels.Count-1 do
  begin
    X := X + stbMensagens.Panels[J].Width;

    if PosicaoMouse.X < X then
    begin
      Painel := J;
      Break;
    end;
  end;

  if Painel = -1 then
    Painel := -1 + stbMensagens.Panels.Count;

  ShowMessage('Você clicou no painel: ' +
    IntToStr(Painel))
end;
```

barra de progresso para que não fique deformada dentro do *StatusBar*.

Retorne agora ao evento *OnDrawPanel* e adicione o código da **Listagem 5** ao final do evento, ou seja, mantenha o que já codificamos antes e apenas adicione mais estas linhas. Aqui estamos verificando se estamos pintando o painel *4*, último painel agora, e então alinhamos a nossa *ProgressBar*. Para testarmos se tudo está funcionando, inclua um botão na tela e codifique seu evento *OnClick* conforme a **Listarem 6**. O código é simples, apenas simulamos o progresso da barra preenchendo-a de *0* a *100* em um laço *for*. (**Figura 10**)

Interagindo com o usuário

Por fim faremos a última modificação, veremos qual painel o nosso usuário clicou. Isso pode ser feito diretamente no evento *OnClick* da *StatusBar*, portanto digite o código da **Listagem 7** e vamos as explicações.

A primeira coisa que fazemos é declarar algumas variáveis para nos auxiliar na captura do painel que o usuário clicou. Em seguida guardamos a posição no mouse em relação a ela e a passamos para o método *ScreenToClient*, que retorna a posição do cursor em relação ao *StatusBar* e o formulário. Para descobrir em cima de qual painel estamos, montamos um laço *for* que fará a comparação das coordenadas do mouse em relação a posição e largura do painel. Depois somamos *-1* a quantidade de painéis e descobrimos exatamente onde estamos, então é só mostrar a mensagem usando um *ShowMessage* (**Figura 11**).

Conclusão

Nesse artigo vimos os principais aspectos da barra de status, *StatusBar*, componente da VCL. Vimos que há diversas possibilidades de personalizar um dos componentes mais interessantes do Delphi. Não há limites para o desenvolvimento, basta muito estudo, dedicação e criatividade. Desta forma é possível extrair o que existe de melhor na linguagem.

Lembre-se que o limite é a criatividade. Boa sorte e até a próxima. ●

Seja um assinante Gold

Com a assinatura Gold você já pode consultar onlin
todos os artigos publicados na sua revista desde a edição nº 1

DevMedia GROUP

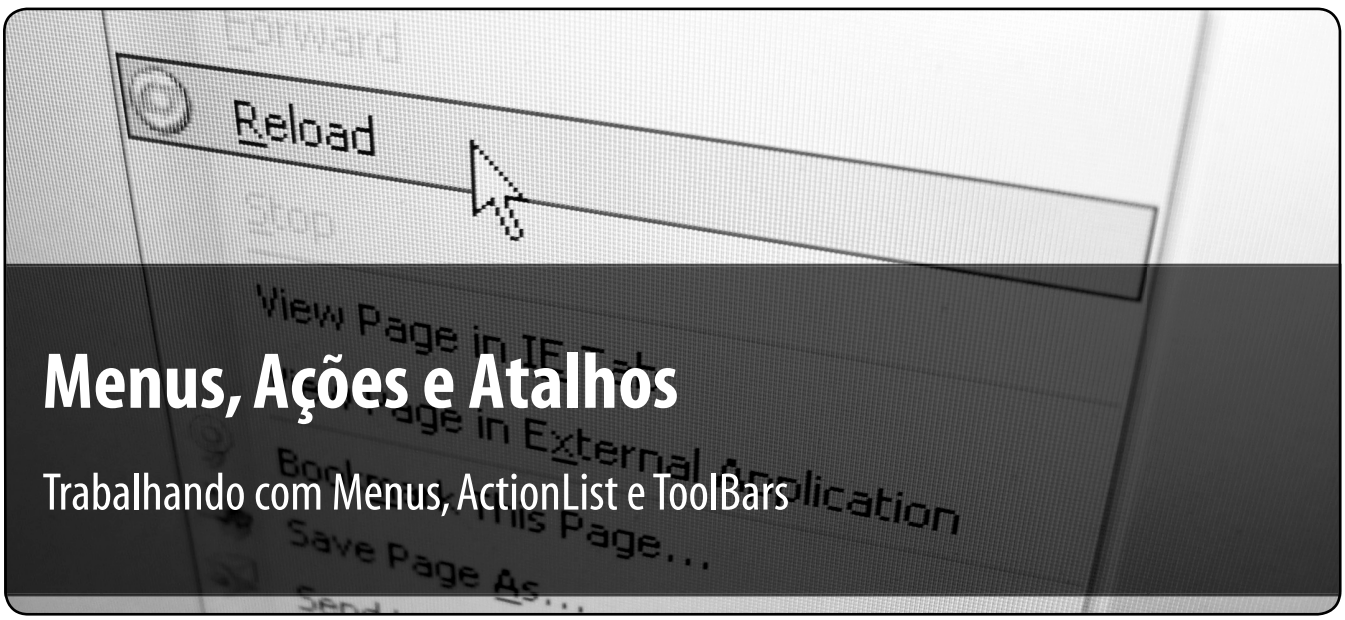
Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine

Nesta seção você encontra artigos para iniciantes na linguagem Delphi



Menus, Ações e Atalhos

Trabalhando com Menus, ActionList e ToolBars

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da Equipe Editorial ClubeDelphi.

Edinei Daniel Steffen
(edineidaniel@gmail.com)
é técnico e Bacharelando em Sistemas de Informação, Analista e Desenvolvedor de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL.

Dificilmente encontramos hoje um sistema que envolva cadastros onde não exista a presença de um menu principal para o acesso aos formulários, bem como os botões de atalho para acesso rápido às janelas. Geralmente estes atalhos estão localizados logo abaixo do menu principal, na forma de pequenos botões que possuem imagens relacionado a ações que o botão executará.

Iremos ver neste artigo um forma simples e eficaz de se criar *Menus*, *ToolBars* e *AcitonList*. Usaremos os componentes *MainMenu*, *ToolBar* e *ActionList*, componentes estes que fazem parte da ferramenta Delphi desde suas primeiras versões.

Partiremos de uma análise do problema com a proposta de desenvolver um sistema para o controle de contas, lembrando que o objetivo deste artigo será o uso e a implementação dos componentes citados anteriormente e não o desenvolvimento e análise do sistema e funções em geral.

Criando a aplicação

Iremos utilizar para este artigo a versão do Delphi 7, fique a vontade para o uso de qualquer outra versão para dar início ao desenvolvimento do protótipo de sistema.

No menu *File|New>Application* crie uma nova aplicação, altere a propriedade *Name* do formulário inicial para "frmPrincipal" e a propriedade *Caption* para "Controle de Movimentações em Contas". Salve a *Unit* principal do projeto como "uPrincipal.pas" e o projeto salve como "Contas.dpr". Em seguida crie outros dois formulários pela opção *File|New>Form* e configure-os conforme a seguir:

• *Form1*: altere a propriedade *Name* para "frmContas" e *Caption* para "Cadastro de Contas". Salve a *Unit* com o nome de "uContas.pas";

• *Form2*: *Name* igual a "frmLancamentos" e *Caption* como "Lançamento de Contas". Salve a *Unit* com o nome de "uLancamentos.pas";

Inicialmente vou dar uma definição resumida dos principais componentes que serão utilizados neste projeto.

MainMenu: Presente na paleta *Standard* e é responsável pela criação e modelagem dos menu;

ToolBar: Componente da paleta *Win32*. Podemos criar barras de ferramentas com botões de acesso rápido as funções de nosso sistema;

ActionList: Esse componente é usado para agrupar em um só lugar várias ações. Não é visível em tempo de execução e pode ser encontrado na paleta *Standard*;

Agora daremos início a real utilização dos componentes de forma bem especificada. Arraste um componente *MainMenu* da paleta *Standard* para o formulário principal. O componente poderá ser visualizado no formulário conforme a **Figura 1.**

Agora faremos a inclusão dos itens de menu. Dê um duplo clique no componente *MainMenu* e observe que uma pequena janela é aberta (**Figura 2)**.

A criação de menus no Delphi é muito simples. Note que no canto esquerdo superior da janela que se abrir está um item em azul, como se estivesse selecionado. Como ainda não adicionamos nenhum item de menu, basta iniciar o processo de digitação e automaticamente o *Object Inspector* se abrirá.

Cada item de menu possui suas propriedades, tais como *Caption*, *Name* e *ShortCut* ("tecla de atalho"). Nesse primeiro menu digite "Arquivo". Automaticamente a propriedade *Name* é atualizada. O nome é gerado pelo Delphi, mas nada impede que se altere caso queira.

Após a criação do menu *Arquivo* o Delphi "pula" para a próxima linha do *MainMenu* nos possibilitando a criação de novos itens. Abaixo de *Arquivo* insira dois novos itens chamando-os de "Novo" e "Abrir", respectivamente. Para inserir um separador entre Abrir e o próximo crie um novo item e digite um traço "-" na propriedade *Caption*. Em seguida insira o menu "Sair". Nosso menu deverá se parecer com a **Figura 3**.

> **Nota**: Para inserir um novo item basta navegar com a seta para baixo do teclado até o último item, ou clicar com o mouse.

Contudo existem ainda possibilidades de adicionar sub-menus ao menu principal. Selecione o meu *Abrir* e dê um clique de direita sobre ele e por último escolha a opção *Create SubMenu*. Agora podemos incluir sub-menus ao item *Abrir*. Inclua dois novos itens: "Imagem" e "Texto". Veja a **Figura 4**.

Depois de desenhado, o menu está pronto para receber ações e essas ações chamamos de eventos. Todo item de menu pode ter eventos associados. Para exemplificarmos clique duas vezes no item de menu *Novo* e seu evento *OnClick* digite o código a seguir.

```
ShowMessage('DevMedia - Clube Delphi!!!');
```

Aqui estamos chamando apenas uma mensagem, porém é perfeitamente possível incluir rotinas e/ou chamadas a outros formulários.

Configurando o ActionList

Um *ActionList*, como o próprio nome sugere, é um componente cujo seu maior legado é agrupar ações. Traduzindo ao pé da letra seria uma *Lista de Ações*.

Deixaremos o menu de lado por um instante e vamos agora fazer a utilização e implementação do componente *ActionList*, ou seja, vamos montar a nossa lista de ações.

Digamos que teremos as seguintes ações envolvidas com nosso sistema:

- *actSair:* Será responsável pelo encerramento do sistema;
- *actFrmContas:* Chamará o formulário para o cadastro de contas;
- *actFrmLancamentos:* Chamará o formulário para o lançamento de contas;
- *actRelatorioLanc:* Responsável pela apresentação do relatório de lançamentos.

Após colocado o componente *ActionList* no formulário de menu, o primeiro passo é criar as ações. Com um duplo clique

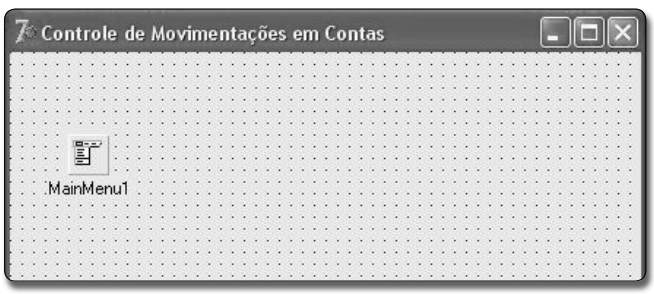


Figura 1. Utilização do componente MainMenu

Figura 3. Menu Arquivo em desenvolvimento

Figura 2. Estrutura de desenho do componente MainMenu

Figura 4. Criação de sub-menus

sobre o componente, será apresentado e editor de ações conforme **Figura 5**.

Para criar uma nova ação basta clicar sobre o ícone *New Action*. Criada a *Action* precisamos configurá-la no *Object Inspetor*. Para isso selecione-a e pressione *F11*. Configure a propriedade *Caption* como "Sair do Sistema" e seu atalho em *ShortCut* deverá ficar como *Ctrl + S*. Em seguida mude para aba *Events* e clique duas vezes no evento *OnExecute* desta ação. Digite o código a seguir:

```
if Application.MessageBox
  ('Deseja Sair do Sistema?', 'Saindo...',
  MB_YESNO+MB_ICONQUESTION)= IDYes then
  Close;
```

Repita o processo anterior e insira uma nova ação. Seu Caption será "Contas" e seu *Name* "actFrmContas". Escolha a combinação *Ctrl + Alt + C* para a propriedade *ShortCut*. Por fim codifique seu evento *OnExecute* conforme a seguir:

```
frmContas.ShowModal;
```

A próxima ação a ser criada terá o *Name* "actFrmLancamentos" e *Caption* "Lançamentos". Escolheremos *Ctrl + Alt + L* para o *ShortCut*. O código de seu evento será como segue:

```
frmLancamentos.ShowModal;
```

Nossa última ação terá o Name igual a "actRelatorioLanc" e *Caption* igual a "Relatório de Contas". Modifique o *ShortCut* de forma que receba as teclas *Ctrl + Alt + R*. Seu evento *OnExecute* terá:

```
ShowMessage(
 'Relatório de Lançamentos em Construção!');
```

A **Figura 6** demonstra as ações criadas e configuradas.

Um vez criadas e configuradas as ações, estas ficam disponíveis para que possamos associá-las a botões e menus. Abra o nosso *MainMenu* criado anteriormente, navegue até o item *Sair*, selecione-o e em seguida pressione *F11* para ver suas propriedades no *Object Inspector*. Clique na propriedade *Action* e note que podemos ver todas as ações criadas no objeto *ActionList*. Pois bem, selecione a ação *actSair* para o item *Sair*.

Agora faremos a criação de um novo menu. Clique no espaço em branco ao lado direito do menu *Arquivo* para e digite "Cadastro". Adicione dois novos itens e associe a propriedade *Action* de cada um as ações *actContas* e *actLancamentos*, respectivamente.

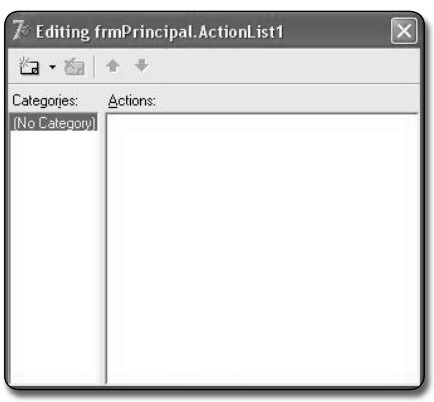


Figura 5. Editor de propriedade do ActionList

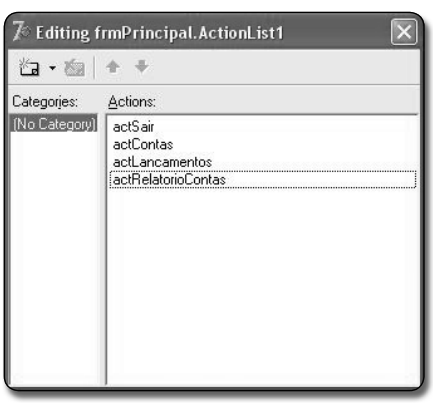


Figura 6. Lista de ações criadas

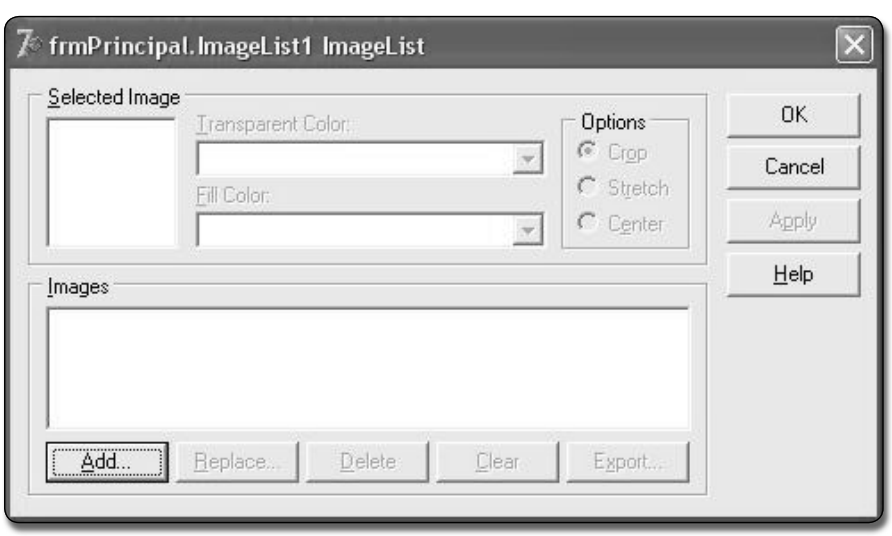


Figura 7. Editor de propriedades do ImageList

Nota do DevMan

Note que no momento em que a ação é relacionada ao menu, o item recupera automaticamente as características da ação usada.

Trabalhando com ToolBars

Vamos incrementar ainda mais nosso exemplo usando uma *ToolBar*. Para tanto localize–o na paleta Win32 e arraste-o para o formulário. Automaticamente o componente se posicionará logo abaixo do menu.

Uma *ToolBar* pode conter *Botões, Edit's, Label's* etc, porém é muito mais comum se encontrar apenas botões. Normalmente esse botões executam a mesma ação que um item de menu em específico, ou seja, um atalho. Encontramos exemplos a todo instante no dia-a-dia. No Word, por exemplo, encontramos o botão *Copiar* na barra de ferramentas e o item *Copiar* no menu *Editar*. Os atalhos na *ToolBar* facilitam a vida do usuário final de agilizam certas ações do sistema.

Em nosso programa exemplo, partiremos do pré-suposto que as telas mais utilizadas são "Lançamento de Contas" e o "Relatório de Lançamentos", portanto criaremos uma *ToolBar* com dois botões para responder a estas ações. Para criarmos botões na *ToolBar* basta clicar com o botão direito nela e selecionar *New Button*. E caso sejam necessários separadores, clique novamente com o direito e clique em *New Separator*.

Da mesma forma que fizemos a associação de ações aos itens de menu, faremos aos botões da barra de ferramentas que criamos. Apenas selecione o botão na barra e na propriedade *Action* selcione

a ação que deseja. Com isso não existe a necessidade de codificar novamente os botões, pois os códigos estão atrelados as ações, que por sua vez são utilizadas por diferentes componentes no formulário.

Em princípio o sistema, configuração e utilização dos componentes propostos está pronto para ser utilizado. Para tornar nosso software mais apresentável e visualmente mais profissional vamos utilizar imagens nos botões para que possamos representar as ações de cada um.

O *ToolBar* nos permite utilizar os recursos de um outro bom e velho componente da VCL, o *ImageList*. O *ImageList* é um componente que armazena imagens nos formatos .bmp e .ico.

Inclua um componente *ImageList* no formulário e clique duas vezes nele. Em seu editor de propriedades (**Figura 7**) clique no botão *Add* para inserir novas imagens.

Insira quatro imagens .bmp ou .ico no com editor de propriedades e confirme. Feito isso faremos o "link" do objeto com a imagem. E isso é muito fácil de se fazer. Basta clicar no objeto, em nosso caso em um dos botões da *ToolBar*, selecionar a propriedade *ImageIndex* e informar qual o índice da imagem que desejamos usar. No caso dos botões e menus que são associados as ações no *ActionList*, não é necessária esta configuração, já que a própria *Action* se encarregará de atualizar seu controle, no caso itens de menus e botões.

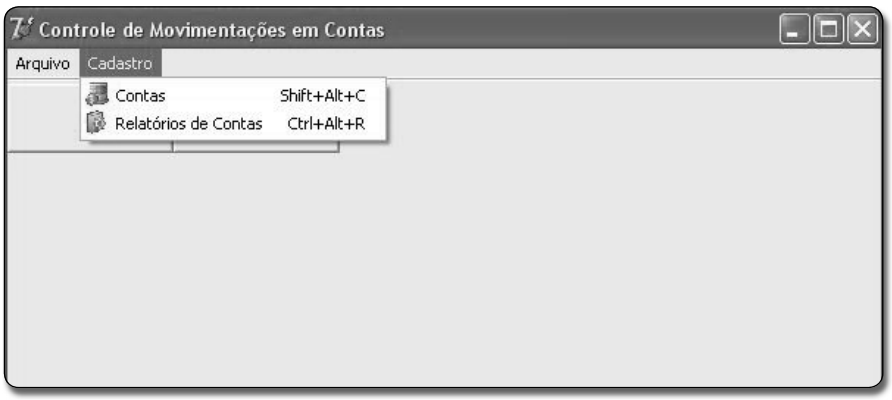


Figura 8. Imagem do sistema funcionando

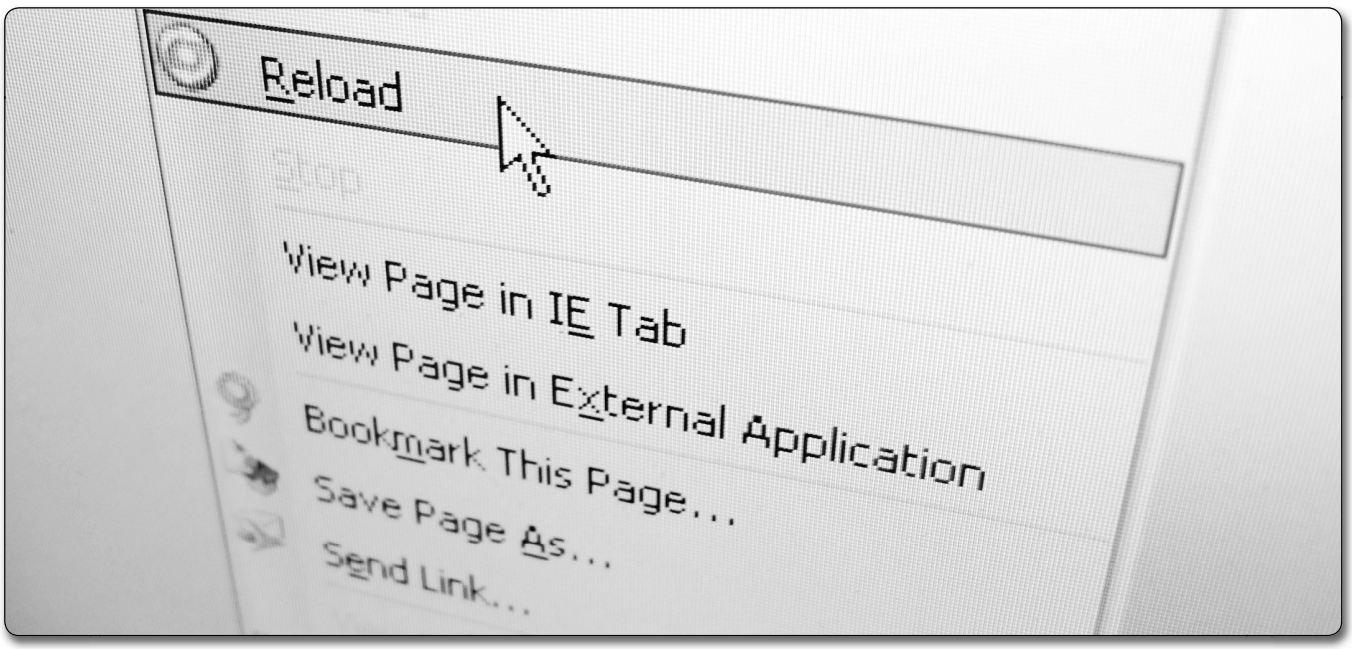
Agora que temos a lista de imagens e as devidas associações, temos que relacionar os componentes *MainMenu*, *ToolBar* e a *ActionList* ao componente *ImageList*. Isso não é problema, pois os três componentes já prevêem o uso deste tipo de controle, portanto selecione qualquer um dos componentes e na propriedade Imagens selecione o *ImageList1*.

Depois de todas estas configurações compile e execute o programa. Veja como ficou interessante o nosso sistema. Temos barras de ferramentas, imagens, menus etc. (**Figura 8**)

Conclusão

Neste artigo, foram abordadas as principais funcionalidades dos componentes *MainMenu*, *ToolBar* e *ActionList*. Vejo que o mais interessante é a *ActionList*, pois permite que você defina e codifique as mais variadas ações para seu sistema, sendo possível atrelar quaisquer ações da *ActionList* com vários componentes visuais do Delphi, por meio da propriedade *Action*. Com isso, você também poderá reutilizar códigos, pois uma ação pode ser relacionada com mais de um componente, não sendo necessário codificar novamente sua funcionalidade.

Muito bem, agora você pode incrementar seus sistemas com as funcionalidades apresentadas nesse artigo e conforme sua criatividade poderá também criar outras funcionalidades aplicáveis em outros procedimentos de seus sistemas. Um grande abraço e até o próximo artigo. ●



Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Orientação a Objetos no Delphi for PHP

Como aplicar conceitos de POO em aplicações PHP?

Rodrigo Carreiro Mourão
(r_c_mourao@hotmail.com)
Consultor da TDS Tecnologia – RJ atuando na área de desenvolvimento de projetos Orientados a Objetos, Design Patterns, MVC. BDS2006 Win32 Product Certified. Instrutor de treinamentos oficiais CodeGear Delphi for Win32, Delphi for PHP, Delphi .Net. Palestrante do Borland Conference 2007.

Sem dúvida nenhuma POO é assunto para vários e vários capítulos de discussão. Nesse artigo veremos como aplicar e trabalhar com Orientação a Objetos em linguagem PHP com a nova ferramenta Delphi for PHP. Com toda certeza poderemos tirar bom proveito dos recursos do novo IDE e desenvolver aplicativos para *Web* cada vez mais robustos e flexíveis.

A POO é o alicerce da programação como um todo, e com ela é possível desenvolver os mais variados aplicativos com diversas vantagens. Essas vantagens podem ser chamadas de *Pilares*, que são ao todo quatro: *Polimorfismo, Herança, Abstração* e *Encapsulamento*.

Veremos os principais aspectos da Orientação a Objetos e como aplicá-los em um *website* escrito em PHP de maneira clara e objetiva. Para entendermos bem todo o conjunto de informações referentes a POO, farei breves comparações com o Delphi para que possamos nos situar e entender melhor como tudo funciona.

Os Pilares da Orientação a Objetos

Toda linguagem dita orientada a objetos deve estar firmada no que chamamos de *Pilares* (**Figura 1**), como dito anteriormente. Estes conceitos não estavam totalmente disponíveis até a versão 4 do PHP. Com o lançamento da versão 5, o modelo orientado a objetos da linguagem foi totalmente reescrito. Em relação a OO não seria exagero dizer que era uma outra linguagem e bastante difícil de lidar. Quem "tentou" programar OO no PHP 4 sabe do que estou falando.

Todos os pilares citados anteriormente estão presentes na versão 5 o que nos permite desenvolver aplicações *Web* em cima desta filosofia. Um modelo orientado a objetos gira em torno de classes e objetos. Como mencionado, OO é um assunto polêmico e muito extenso, portanto não serão abordados aqui todos os seus conceitos a fundo. É altamente recomendado um estudo aprofundado dos conceitos básicos da POO para que se possa entender perfei-

tamente toda a estrutura de programação nesses moldes. Duas boas referências para os iniciantes estão na revista ClubeDelphi 91 com dois excelentes artigos dos autores Adriano Santos e Maikel Sheid.

Contudo para que se possa situar, as classes seriam como a planta de um edifício com suas definições de altura, largura, quantidades de andares, elevadores etc., ou seja, o planejamento do edifício, algo ainda no papel, não tangível. Já o objeto seria o próprio edifício construído, concreto e palpável.

Todo objeto possui suas características que chamamos de *propriedades*. Possui também comportamentos que chamamos de *métodos*. Tudo isso definido na classe ao qual este objeto pertence. A vantagem da OO em relação a outras filosofias de programação é justamente a possibilidades de criar objetos passíveis de serem reaproveitados reduzindo assim o tempo de desenvolvimento. Outra vantagem é a de manter os dados (*propriedades*) e regras de negócios (*métodos*) unidos numa mesma estrutura (*objeto*).

Uma vez tendo estes conceitos em mente, basta aplicá-los a linguagem em que iremos trabalhar, no nosso caso PHP. Bem, até agora está tudo muito bom, mas na prática, como isso se aplica? Esta é a pergunta que não quer calar. Primeiro vamos à sintaxe OO no PHP.

Criando e entendendo as classes

Para criarmos uma classe utilizamos, como no Delphi, a palavra reservada *class*, porém a sintaxe é um pouco diferente, como podemos ver a seguir:

```
class Pessoa{}
```

Como podemos ver, não temos aqui a presença do operador *type* como no Delphi. As propriedades e métodos desta classe devem ser definidos entre as chaves que a definem. Tudo que for colocado neste escopo pertence a ela:

```
class Pessoa{
  var $nome;
  var $email;
}
```

Entendendo os objetos

Temos agora nossa classe que possui a definição do que será nosso objeto propriamente dito. Precisamos nesse momento criar esses objetos, alocá-los em memória e torná-los reais. Para instanciarmos uma classe no PHP utilizamos a palavra reservada *new*. Neste ponto não custa lembrar que o PHP não é fortemente tipado como é o Win32 e mais ainda ele é *Case Sensitive*, ou seja, difere maiúsculas de minúsculas, portanto muita atenção na hora que definir suas classes e as variáveis que irão receber seus objetos. Para criarmos um da classe *Pessoa*, definida anteriormente, bastaria atribuir a uma variável o resultado da função *new* como a seguir:

```
$joao = new Pessoa;
```

Neste momento temos em *$joao* um objeto com as definições da nossa classe, com isso é fato que *$joao* tem um nome e possui um e-mail, ambos aptos a receberem valores.

Listagem 1. Inclusão do construtor da classe

```
class Pessoa{
  var $nome;
  var $email;
  function __construct($nm, $em){
  }
}
```

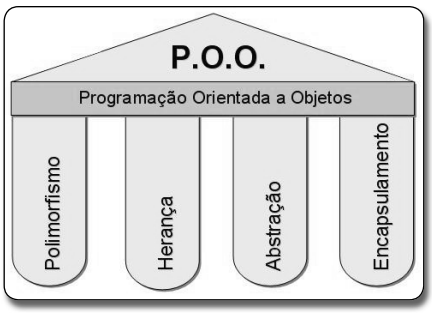


Figura 1. Pilares da programação orientada a objetos

Construtores

Até o PHP 4, para definirmos um construtor para nossa classe, bastava criar um método com o mesmo nome da classe que este seria invocado quando o objeto fosse construído. Isso ainda funciona com o PHP 5, porém agora temos um método construtor exclusivo para nossas classes e recomenda-se utilizá-lo uma vez que o conceito anterior só foi mantido por questões de compatibilidade.

No PHP 5 quando queremos determinar um construtor para nossa classe basta incluirmos na mesma o método *__construct()*. Este método será invocado sempre que um novo objeto desta classe for instanciado, com isso podemos definir regras para nossa classe como, por exemplo, só permitir que se crie uma nova pessoa se um nome e e-mail forem informados na hora de instanciar o objeto (**Listagem 1**).

Neste momento o código em que criamos o objeto *$joao* já não funciona mais, pois agora o construtor da classe ao qual *$joao* pertence pede dois parâmetros para ser invocado. O correto pode ser visto no código que segue:

```
$joao = new Pessoa('João das couves',
  'joao@joao.com.br');
```

Esta seria a forma correta de instanciarmos nosso objeto, pois agora seu construtor pede dois parâmetros e a palavra reservada *new* invoca o construtor, assim como no Delphi o *Classe.Create()* também invoca o método construtor.

Destrutores

Esta é uma novidade no PHP 5. Agora podemos definir métodos que são invocados quando um objeto é destruído: o destrutor. Assim como no caso dos construtores há um método exclusivo para ele. Basta inserirmos o método *__destruct* na definição de nossa classe.

Este método será invocado quando o objeto for destruído e isto acontece em duas situações no PHP. A primeira quando a referência ao objeto e destruída (*$joao = null*) e a outra é quando a requisição chega ao final. Como estamos desenvolvendo para *Web*, devemos lembrar que este ambiente é *stateless*, ou seja, não guarda por padrão valores de uma outra requisição, nós é que devemos nos valer de mecanismos para manter esses valores entre as requisições (*Session, Cookies. etc*).

Nota: Perceba que os métodos *constructor* e *destructor* são escritos com dois caracteres undercore no início da palavra. Isso é um padrão da linguagem.

A variável $this

Até agora está tudo muito bem, porém o nosso construtor não está nos valendo de nada, pois os parâmetros a ele passados não estão sendo utilizados, precisamos receber esses parâmetros e repassarmos para as propriedades *$nome* e *$email* e é ai que um conceito precisa ficar claro, pois isso difere muito do conceito que conhecemos no Delphi.

No Delphi, quando nos quereremos acessar uma propriedade de dentro da própria classe, basta digitar o nome desta ou mesmo chamar um método, pois o compilador sabe que aquela propriedade ou método pertence àquela classe, no PHP isso não acontece. Como PHP é um linguagem interpretada, não há necessidade de se declarar variáveis, muito menos declarar métodos e depois implementá-los. Um outro detalhe neste caso é que as variáveis criadas dentro de uma função têm como escopo a própria função, então observe o código da **Listagem 2**.

Como, neste caso, o interpretador irá entender que você não está criando uma variável e sim acessando a propriedade definida na classe? Não vai! O que ele vai fazer aqui e exatamente criar duas variáveis cujo escopo é a função e então atribuir a elas os valores passados como parâmetro, nada será feito com as propriedades do objeto, que ficarão intactas.

Para resolver este problema, o PHP traz em sua sintaxe OO a variável *$this*. Ela é usada sempre que você quiser acessar métodos e propriedades de dentro da própria classe, um conceito bem diferente da programação Win32 no Delphi. Seria redundante você escrever o código adiante para trocar o *Caption* de um *Label* em um formulário:

```
Self.Label1.Caption := 'Rodrigo Mourão';
```

Já no PHP isso não é redundante e sim necessário, porém no lugar do *Self* utilizamos o *$this*, mas isso não significa que o PHP não tenha o seu *Self*. O código correto para nosso construtor podemos ver na **Listagem 3**.

Agora sim o interpretador tem ciência de que você não esta criando duas variáveis novas e sim acessando propriedades do objeto onde o construtor está sendo chamado.

Listagem 2. Declaração das propriedades da classe

```
class Pessoa{
  var $nome;
  var $email;
  function __construct($nm, $em){
   $nome = $nm;
    $email = $em;
  }
}
```

Listagem 3. Uso do $this na atribuição dos valores das propriedades

```
class Pessoa{
  var $nome;
  var $email;
  function __construct($nm, $em){
    $this->nome = $nm;
    $this->email = $em;
  }
}
```

Listagem 4. Uso de private e public em propriedades e funções

```
class Pessoa{
  private $nome;
  private $email;
  public function __construct($nm, $em){
    $this->nome = $nm;
    $this->email = $em;
  }
}
```

Listagem 5. Aplicação de regras de negócio na classe

```
class Pessoa{
  private $nome;
  private $email;
  function __construct($nm, $em){
  $this->nome = strtoupper($nm);
  $this->email = strtolower($em);
}
```

Escopo de Visibilidade

Você deve ter notado a maneira pela qual criei as propriedades da nossa classe:

```
var $nome;
```

Esta era a maneira utilizada até a versão 4, todas as propriedades e métodos eram públicos por padrão e com isso visíveis de qualquer outra classe externa. A partir da versão 5 foram introduzidos operadores para definir escopo de visibilidade, e como no Delphi, se nenhum escopo for definido o padrão será público.

Agora suas classes podem ter métodos e propriedades privadas("private"), protegidas ("protected") e públicas ("public").

• *Private:* Métodos e propriedades visíveis apenas dentro da própria classe, não sendo passíveis de acessá-los externamente. Nesse caso esses valores são manipulados pela própria classe por métodos que são públicos;

• *Protected:* Métodos e propriedades que são visíveis dentro da própria classe e também das classes que descendem da mesma. No PHP utilizamos este escopo na maioria das vezes para declaramos métodos que serão sobrescritos numa classe descendente, porém não serão acessíveis de fora desta hierarquia;

• *Public:* Visíveis dentro da própria classe, classes descendentes e de qualquer outra classe externa a ela. Geralmente em OO encapsulamos o que varia e disponibilizamos métodos públicos para acessar estas propriedades;

Podemos nos valer desses escopos de visibilidade para adicionar mais uma regra a nossa classe *Pessoa* não permitindo que se acesse suas propriedades externamente e com isso garantir que o *nome* e *e-mail* sejam repassados somente pelo construtor e assim verificarmos se esses valores atendem as nossas especificações. Veja como podemos fazer para que nossas propriedades não sejam visíveis ao mundo externo visualizando o código da **Listagem 4**.

Nota do DevMan

O operador public seria desnecessário aqui visto que quando não informado o escopo padrão é público.

Desta maneira conseguimos validar a entrada de dados nas propriedades. Poderíamos aqui verificar se o e-mail passado é um endereço válido através de uma expressão regular ou mesmo colocar o nome passado pelo parâmetro em letras maiúsculas.

Herança

Um dos motivos que levam muitas pessoas a programarem OO é justamente a possibilidade de se reaproveitar códigos e regras através da herança. Claro que seu uso indiscriminado pode vir a prejudicar o projeto, pois quando se fala em herança fala-se também em acoplamento.

Herdar de uma classe significa possuir tudo aquilo que a classe *Pai* possui e poder acrescentar algo mais que seja pertinente a ela. Ainda no contexto da

nossa classe *Pessoa*, vamos fazer novas modificações no construtor para tratar os valores que serão armazenados em suas propriedades. A regra será nome em letras maiúsculas e e-mail em letras minúsculas (**Listagem 5**).

Não importa como os dados serão enviados ao construtor. Uma vez recebidos pela classe, ela se encarrega de efetuar os tratamentos antes de repassá-los às propriedades. Esse conceito é chamado de encapsulamento. E com isso criamos nossa regra.

Pensando na classe *Pessoa*, podemos criar duas novas classe *PessoaFisica* e *PessoaJuridica,* ambas com *nome* e *e-mail* e cada uma com uma propriedade exclusiva *CPF* e *CNPJ* respectivamente.

Não criaremos duas novas classes com todos os atributos e depois as mesmas regras de negócio para elas. É aí que entra a herança, uma vez que a classe *Pessoa* já detém algum dos atributos e as regras.

O que faremos aqui é simplesmente fazer com que as classes *PessoaFisica* e *PessoaJuridica* herdem de *Pessoa* e, conseqüentemente, possua as mesmas propriedades, métodos e regras de negócios. Feito isso acrescentaremos as propriedades pertinentes a cada uma exclusivamente. Essa é uma das vantagens de se trabalhar com herança, o que justifica seu uso em um modelo OO é justamente a possibilidade de se implementar o polimorfismo, uma vez que com herança temos acoplamento e nem sempre é isso que queremos em nossos códigos.

Entendido o que será feito basta agora fazê-lo em PHP. O PHP utiliza a palavra reservada *extends* para indicar o uso de herança no código. Na **Listagem 6** podemos ver a definição das classes *Pessoa*, *PessoaFisica* e *PessoaJuridica*.

> **Nota**: O PHP 5 ainda não possui suporte a herança múltipla, nesse caso é necessário fazer uso de *Interfaces*, o que está fora do escopo deste artigo.

Aparentemente nossas classes estão prontas para uso, mas há um detalhe a se entender. Da maneira que está se o usuário que quiser instanciar umas das classes (*PessoaFisica)* ou (*PessoaJuridica),* deverá informar agora três parâmetros ao construtor, até aí tudo bem. Porém se reparar bem, somente o *CPF* ou *CNPJ* está sendo repassado para a classe, *nome* e *e-mail* não. O problema é que não podemos utilizar a variável *$this* para elas, pois essas propriedades foram declaradas como *private* ("privadas"), ou seja, mesmo pertencendo ao mesmo *Objeto* só serão acessíveis pela classe onde foram declaradas inicialmente, neste caso *Pessoa*.

Neste momento você deve estar pensando: "Simples vamos alterar o escopo para *Protected*, assim elas serão visíveis também nas classes descendentes". Em parte, se você pensou nesta solução, não está de todo errado, porém se fizermos isso estaremos burlando a regras que nós estabelecemos anteriormente. Isso significa que as regras criadas para colocar maiúsculas e minúsculas em nome e e-mail não se aplicarão a *PessoaFisica* e *PessoaJuridica*, pois foram implementadas no construtor da classe *Pessoa*, sendo assim temos que de alguma maneira invocar o construtor de *Pessoa* quando o construtor de *PessoaFisica* ou *PessoaJuridica* for chamado. Lembra *inherited* do Delphi? Pois bem.

Vale lembrar que, no PHP, se nenhum método construtor for declarado na classe *Filha*, o construtor do *Pai* é automaticamente chamado. Se sobrescrevermos o construtor ou qualquer outro método nos *Filhos,* o método na classe *Pai* deve ser invocado explicitamente.

A solução para o impasse anterior é justamente utilizar o operador *Parent*. Sua função é exatamente invocar um método na classe imediatamente acima na estrutura hierárquica de suas classes e com isso garantir que as regras e códigos sejam executados. Observe agora como ficaram nossas classes com esta implementação. (**Listagem 7**)

Agora sim podemos ter certeza que nossa regra de negócio será implementada em ambas as classes descendentes. Repare que invocamos o construtor de *Pessoa* e repassamos os parâmetros que o mesmo necessita, *nome* e *e-mail*. O terceiro parâmetro é atribuído a própria

Listagem 6. Declaração das classes Pessoa, PessoaFisica e PessoaJuridica

```
class Pessoa{
  private $nome;
  private $email;
  function __construct($nm, $em){
    $this->nome = strtoupper($nm);
    $this->email = strtolower($em);
  }
class PessoaFisica extends Pessoa{private $cpf;
  function __construct($nm, $em, $cpf){
   $this->cpf = $cpf;
  }
}
class PessoaJuridica extends Pessoa{private $cnpj;
  function __construct($nm, $em, $cnpj){
   $this->cnpj = $cnpj;
  }
}
```

Listagem 7. Implentação do Parent

```
class Pessoa{
  private $nome;
  private $email;
  function __construct($nm, $em){
    $this->nome = strtoupper($nm);
    $this->email = strtolower($em);
  }
}
class PessoaFisica extends Pessoa{private $cpf;
   function __construct($nm, $em, $cpf){
    parent::__construct($nm, $em);
    $this->cpf = $cpf;
   }
}
class PessoaJuridica extends Pessoa{
  private $cnpj;
    function __construct($nm, $em, $cnpj){
    parent::__construct($nm, $em);
  $this->cnpj = $cnpj;
  }
}
```

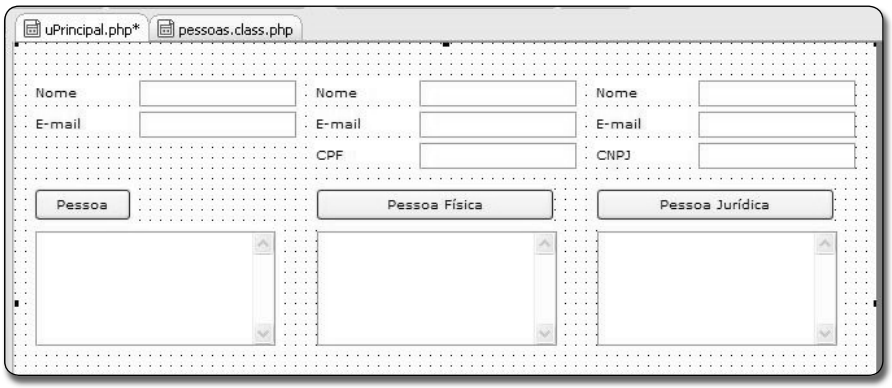


Figura 2. Exemplo de formulário para testes de classes

Listagem 8. Declaração de um formulário no Delphi for PHP

```
<?php
  //Includes
  require_once("vcl/vcl.inc.php");
  use_unit("forms.inc.php");
  use_unit("extctrls.inc.php");
  use_unit("stdctrls.inc.php");

  //Class definition
  class Unit3 extends Page{
    public $Button1 = null;
    public $Label1 = null;
    public $Edit1 = null;
  }
  global $application;
  global $Unit3;

  //Creates the form
  $Unit3 = new Unit3($application);

  //Read from resource file
  $Unit3->loadResource(__FILE__);

  //Shows the form
  $Unit3->show();
?>
```

Listagem 9. Código de classes do arquivo pessoas.class.php

```
<?php
  class Pessoa{
    private $nome;
    private $email;
    function __construct($nm, $em){
      $this->nome = strtoupper($nm);
      $this->email = strtolower($em);
    }
    function getnome(){
      return $this->nome;
    }
    function getemail(){
      return $this->email;
    }
  }
  class PessoaFisica extends Pessoa{
    private $cpf;
    function __construct($nm, $em, $cpf){
      parent::__construct($nm, $em);
      $this->cpf = $cpf;
    }
    function getcpf(){
      return $this->cpf;
    }
  }
  class PessoaJuridica extends Pessoa{
    private $cnpj;
    function __construct($nm, $em, $cnpj){
      parent::__construct($nm, $em);
      $this->cnpj = $cnpj;
    }
    function getcnpj(){
      return $this->cnpj;
    }
  }
?>
```

classe que o detém, mas lembre-se, o objeto criado em memória é um só, uma pessoa física com nome, e-mail e CPF ou uma pessoa jurídica com nome, e-mail e CNPJ, porém compartilhando regras em comum que foram implementadas em uma classe ascendente.

Acredito que agora o código da **Listagem 8** faça mais sentido para você. Sabemos agora que quando adicionamos um novo formulário em um projeto no Delphi for PHP estamos na verdade adicionando uma classe *Unit1*, *Unit2*, *Unit3* etc que herda da classe *Page* e que possui tantas propriedades quantos componentes adicionarmos no formulário. Se verificar no código em seguida, veremos que *Unit3* está sendo instanciada e repassada para a variável *$Unit3*, ou seja, criando um novo objeto.

```
$Unit3 = new Unit3($application);
```

O construtor do formulário, como no Delphi, pede um parâmetro que na verdade é o *Owner* ("proprietário"). Aqui está sendo passado a variável *global $application*. Agora se entende o porquê de se colocar o *$this* sempre que queremos manipular um componente via código no Delphi for PHP. Todos os componentes são propriedades da classe por isso usamos o *$this*.

Desenvolvendo uma aplicação teste

Bem, nos resta testar o nosso modelo para ver se tudo está funcionando conforme o planejado. Abra o Delphi for PHP, vá em *File|New>Application* e salve a *Unit* principal como "uPrincipal.php". Em seguida salve o projeto como "poo.phprj". Em seguida crie uma nova *Unit* em *File|New>Unit* onde faremos a codificação de nossas classes. Salve-a como "pessoas.class.php" no mesmo diretório da aplicação e digite o código da **Listagem 9**.

Tudo que vimos até agora foi acrescentado a nossa *Unit pessoa.class.php*, com exceção dos métodos *getnome(), getemail(), getcpf()* e *getcnpj*. Isso porque nossas propriedades são privadas, então precisamos de um método público para auxiliar no retorno do conteúdo das mesmas para podermos nos certificar que nossas regras estão funcionando bem.

Agora no formulário principal vamos colocar alguns controles para que possamos interagir com nossas classes. Desenhe um formulário como no exemplo mostrado da **Figura 2**.

Insira oito componentes *Edit* da paleta *Standard*. Modifique o *Name* de cada um para "EdtNomeP", "EdtEmailP", "EdtNomePF", "EdtEmailPF", "EdtCPF", "EdtNomePJ", "EdtEmailPJ" e "EdtCNPJ". Coloque também três *Button's* com os *Captions* conforme a **Figura 2** e *Name's* "BtnPessoa", "BtnPessoaPF" e "BtnPessoaPJ". Abaixo dos botões vão três componentes *Memo* com os *Name's* "mmP", "mmPF" e "mmPJ", da esquerda para a direita.

O que temos são *Edit's* para inserirmos os valores para as propriedades de cada classe e um *Memo* para publicar os valores. Como nossas classes foram definidas em outra *Unit*, precisamos fazer referência a ela em nosso formulário. Para isso use a opção *File>Use Unit* ou pressione *Alt + F12* e selecione *pessoas.class.php*, assim como no Delphi.

O último passo é codificar cada botão do exemplo. Basta clicar duas vezes no botão para que sejamos enviados ao *Code Editor* do Delphi for PHP. Digite o código da **Listagem 10** no botão *BtnPessoa*. Repita esses passos para *BtnPessoaPF* (**Listagem 11**) e *BtnPessoaPJ* (**Listagem 12**).

A rotina é praticamente a mesma para os três casos. Primeiro criamos uma variável para receber a instância da nossa classe. Em seguida no construtor passamos os parâmetros que cada uma delas requer e neste caso passamos o conteúdo dos respectivos *Edit's*. Neste ponto nossa regra já foi aplicada, pois é no construtor que executados nosso código e é ai que entra em cena as funções *getnome(), getemail(), getcpf()* e *getcnpj()*.

São através delas que recuperamos os valores devidamente formatados e publicamos nos controles *Memos* para que possamos verificar o conteúdo das propriedades. Execute a aplicação, preencha os *Edit's* e veja o resultado (**Figura 3**).

Observe que a regra estabelecida em *Pessoa* foi aplicada também em *PessoaFisica* e *PessoaJuridica*. Vale lembrar que qualquer alteração feita na mesma também ira refletir em ambas, por exemplo, a validação do e-mail. Você pode incluir uma rotina para validar o endereço antes de atribuí-lo e fará isso na classe *Pessoa*, esta regra automaticamente testará disponível nas demais.

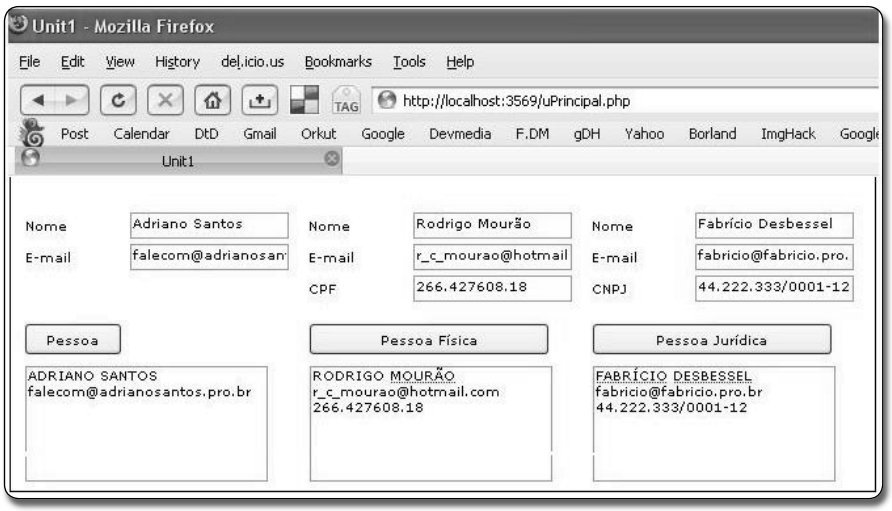


Figura 3. Aplicação em execução, já com os testes das classes

Listagem 10. Código do BtnPessoa

```
function BtnPessoaClick($sender, $params)
{
  $p = new Pessoa($this->EdtNomeP->Text,
    $this->EdtEmailP->Text);
  $this->mmP->Add($p->getnome());
  $this->mmP->Add($p->getemail());
}
```

Listagem 11. Código do BtnPessoaPF

```
function BtnPessoaPFClick($sender, $params)
{
  $f = new PessoaFisica($this->EdtNomePF->Text,
    $this->EdtEmailPF->Text, $this->EdtCPF->Text);
  $this->mmPF->Add($f->getnome());
  $this->mmPF->Add($f->getemail());
  $this->mmPF->Add($f->getcpf());
}
```

Listagem 12. Código do BtnPessoaPJ

```
function BtnPessoaPJClick($sender, $params)
{
  $j = new PessoaJuridica($this->EdtNomePJ->Text,
    $this->EdtEmailPJ->Text, $this->EdtCNPJ->Text);
  $this->mmPJ->Add($j->getnome());
  $this->mmPJ->Add($j->getemail());
  $this->mmPJ->Add($j->getcnpj());
}
```

Conclusão

Muito ainda há o que se falar em relação a OO no Delphi for PHP, mas isso fica para os próximos artigos onde abordaremos tópicos avançados como *polimorfismo*, *classes abstratas*, *métodos* e *propriedades estáticas*, *interfaces* e *Design Patterns*. Espero que esta pequena introdução já lhe ajude nas tarefas simples do seu dia-a-dia de desenvolvedor. OO é mais do que uma "maneira" de se programar, é uma filosofia de desenvolvimento onde temos total controle sobre aquilo que estamos criando.

Olhe a sua volta, enumere cinco linguagens de programação que conhece, agora das cinco quais são Orientadas a Objetos? Veja a importância de tal metodologia. Não se trata de Delphi, PHP, Java, etc. Estamos falando de Orientação a Objetos, algo que é comum a todas essas.

Eu sou Rodrigo Carreiro e pela sua atenção muito obrigado! ●

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Usando páginas de estilo (CSS)

Aprenda como melhorar a aparência de seu site com CSS no Delphi for PHP

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Nas últimas edições da revista ClubeDelphi aprendemos algumas técnicas para o desenvolvimento de aplicações com o novo Delphi for PHP, IDE de desenvolvimento especialmente criado para a linguagem PHP.

Nesse artigo mostrarei como melhorar o *layout* de sua aplicação utilizando o fabuloso recurso de páginas de estilo, CSS (Cascading Style Sheets), no Delphi for PHP. Veremos o que exatamente é CSS, qual sua utilidade e como usá-lo de maneira bastante simples.

Entendendo e usando CSS

O significado do CSS é Cascading Style Sheets ou Folhas de Estilo em Cascata. Se consultarmos o site *wikipedia.org* encontraremos a seguinte definição sobre CSS:

Cascading Style Sheets, ou simplesmente CSS, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Como podemos ver, a principal utilidade do CSS é separar *layout* (parte visual) da programação em si. Desta forma podemos ter um código mais simples e menos confuso. Além disso, podemos facilmente padronizar toda a parte visual de nosso site vinculando-o a um arquivo externo. Em uma eventual mudança de layout, tal como troca de fonte, cores, estilo de texto, bastando alterar o código CSS contido no arquivo externo e reenviá-lo ao site.

Há ainda aqueles que utilizam o CSS para criar propagandas para o *website* no estilo *pop-up*, porém sem a possibilidade de ser bloqueada por bloqueadores de *pop-up* presente na maioria dos *Browsers* hoje em dia.

A sintaxe do CSS

O CSS nada mais é um conjunto de regras previstas em um arquivo texto com a

extensão .css. O arquivo pode ser digitado em qualquer editor de textos, incluindo o *Bloco de Notas* do Windows. Veja um exemplo de CSS na **Listagem 1**.

Basicamente estamos definindo três classes que mais tarde serão utilizadas pelos nossos controles no Delphi for PHP. A primeira classe, *.data*, fará a formatação da data principal do sistema colocando-a na cor vermelha ("#FF0000"), fonte igual a *Trebuchet MS* e alinhamento a direita além de definir seu tamanho ("font-size") e cor de fundo ("background-color"). No mesmo conceito seguem duas outras classes para formatação de *Títulos*(".titulos") e *Botões*(".botoes").

Um exemplo bem simples do uso de CSS pode ser visto no link www.doiscliques.com/pub/web/padroes2/tela.htm. Experimente clicar nos *links Usar Css1* e *Usar Css2*. Note que as páginas contêm o mesmo conteúdo, porém cada página usa um arquivo CSS diferente.

Em nosso exemplo faremos a criação de uma página de cadastro com alguns controles. Cada controle será vinculado a uma classe no CSS e terá seu *layout* alterado tanto na IDE do Delphi for PHP quanto nos *browsers* Internet Explorer e Firefox. Vejamos como proceder para a criação do exemplo.

Nota do DevMan

Em alguns casos e dependendo da versão do browser o código CSS pode ser levemente mau interpretado ou até mesmo não ser exibido. É altamente recomendável testar o site em diferentes browsers, como por exemplo Internet Explorer, Firebox e Opera.

Criando o exemplo

Abra o Delphi for PHP e crie uma nova aplicação selecionando o menu *File|New>Application*. Salve a *Unit* criada como "index.php" e o projeto como "CSS.phprj". Ajuste a largura da página para *700 px* na propriedade *Width* e logo em seguida desenhe uma tela como na **Figura 1**. Em resumo estamos usando onze *Label's*, sete *Edit's*, dois *Button's*, um *ComboBox*, um *RadioGroup* e um *CheckBox*, todos da paleta *Standard*. Os componentes *ComboBox* e *RadioGroup* possuem a propriedade *Items*, onde é possível incluirmos todos os itens que desejamos.

Listagem 1. Exemplo de código CSS

```
.data{
        font-family: Trebuchet MS;
        text-align: right;
        color: #FF0000;
        font-size: 11px;
        background-color: #fff;
}
.titulos{
        font-family: Trebuchet MS;
        text-align: center;
        color: #fff;
        font-size: 20px;
        background-color: #AE5700;
}

.botoes{
        color: #fff;
        font-size: 11px;
        font-family: Trebuchet MS;
        background-color: #4DA4D2;
        border-width: 1px;
        border-style: dashed;
        border-color: #000;
}
```

Listagem 2. Código para formatação da data atual

```
/* Mostra 3 primeiras letras do dia da semana em
ingles */
$s = date("D");
/* Mostra o Mês em números */
$m = date("n");
$dia = date("d");
$ano = date("Y");
$semana = array("Sun" => "Domingo", "Mon" =>
  "Segunda", "Tue" => "Terça", "Wed" => "Quarta",
  "Thu" => "Quinta", "Fri" => "Sexta", "Sat" => "Sábado");
/* Dias da Semana. */
$mes = array(1 =>"Janeiro", "Fevereiro", "Março",
  "Abril", "Maio", "Junho", "Julho", "Agosto",
  "Setembro", "Outubro", "Novembro", "Dezembro");
/* Meses */
print "São Paulo, $semana[$s], $dia de $mes[$m] de $ano";
```

Logo abaixo do logotipo da empresa, inclua um *Label* e modifique seu *Width* para "678 px" e *Left* para *"10"*. Em seu evento *OnShow* digite o código da **Listagem 2**. No código estamos criando quatro variáveis que receberão, respectivamente, *data*, *dia*, *ano* e *semana*. Trocamos os nomes dos meses e dias em inglês para português. Por fim exibimos a mensagem "São Paulo, " concatenado com o valor das variáveis já implementadas. Veja o resultado da codificação:

```
São Paulo, Quinta, 31 de Janeiro de 2008
```

Não haveria problema algum em formatar cada controle no Delphi for PHP usando para isso o próprio IDE, porém, caso precisássemos alterar um estilo de fonte ou mesmo a cor padrão de todas as páginas, teríamos que abrir fonte a fonte e efetuar a mudança. Em um projeto pequeno isso é muito simples, mas em um grande portal a mudança poderia ser demorada e ainda arriscada a erros e esquecimentos por parte dos desenvolvedores.

Voltando ao exemplo, abra o *Bloco de Notas* do Windows e vamos criar nosso arquivo externo com o layout. Digite o código da **Listagem 3** e vamos as explicações antes de testarmos. Nossas três primeiras regras fazem referência à data principal do sistema, títulos e botões assim como já foi visto anteriormente. Em seguida codificamos um layout padrão para *TextBoxes*, *Labels*, *RadioGroups* e *ComboBoxes*. Note que todas as regras foram definidas usando um ponto em seu nome. Mas outras regras também pode ser definidas para outros elementos dentro da página, como o *body*("corpo da página") e *link's*, como veremos mais adiante.

Salve o arquivo como "estilos.css" no diretório onde salvo a página *index.php*. Para utilizar o CSS em nossa aplicação basta incluir um componente *StyleSheet* da paleta *System*. Agora precisamos selecionar o arquivo externo que possui o *layout* de nossa página usando a propriedade *FileName*.

Agora o que precisamos fazer é vincular campo a campo às regras criadas no arquivo. E é muito fácil, basta clicar no compo-

nente e selecionar a propriedade *Style*. Note que todas as regras criadas aparecem na lista como podemos ver na **Figura 2**.

Modifique a propriedade *Style* de cada componente em tela. Perceba que ao selecionar a regra podemos ver o resultado final diretamente no Delphi for PHP assim como podemos visualizar na **Figura 3**. Agora basta executarmos a aplicação clicando em *Run* ou pressionando *F9* que automaticamente a página será visualizada no *browser* padrão.

Para quem deseja testar o exemplo em mais de um *browser* basta entrar em *Tools|Options>Environment Options*. Em seguida clique em *Add* e localize o arquivo executável do *browser* que deseja adicionar, por exemplo *C:\Arquivos de Programas\Internet Explorer\iexplorer.exe*. Agora para testar basta clicar na seta ao lado dos botões *Run* ou *Run Without Debbuging* e escolher o *browser* (**Figura 4**). Na **Figura 5** podemos ver o resultado final da página que criamos sendo executada no Internet Explorer.

Personalizando Links

Além dos elementos que alteramos, também é possível alterar os *link's* da página, como mencionei anteriormente. Nesse caso criaremos um pequeno menu acima da página principal e faremos com que os *link's* sejam modificados durante o *hover*, ou seja, quando o mouse passa por cima dele. Podemos modificar quatro estágios do link:

- *link*: Estado normal do link;
- *active*: Quando o link está ativado, normalmente ao passar por ele com o teclado;
- *visited*: Link já visitado;
- *hover*: Ao passar o mouse sobre o link;

Inclua na parte superior da página, junto ao logotipo, quatro *Label's* alterando os *Caption's* para "Página principal", "Cadastro", "Notícias do Dia" e "Contato". Agora abra novamente o arquivo *estilos.css* e inclua os códigos da **Listagem 4.** A primeira regra que estamos codificando fará a modificação do *link's* quando em *stand-by*, ou seja, quando ainda não foi selecionado, clicado ou está sob o cursor do mouse. Em seguida modificamos os *link's* visitados, alterando sua cor para vermelho (#FF0000). Por último está a parte mais interessante, quando passamos o mouse no *link*. Ele ficará com as bordas tracejadas ("border-style: dashed"), brancas ("border-color") e cor azul ("background-color"). Para links, não é necessário selecionar o estilo ("Style") nas propriedades do *Label*. O código HTML gerado pelo PHP fará isso automaticamente pra nós. Rode a aplicação e veja o resultado. (**Figura 6**)

Aplicando CSS em outras páginas

Para finalizar o nosso exemplo, vejamos como aplicar CSS a outras páginas da aplicação e como a elas se comportam ao ter o arquivo externo alterado. Crie uma nova página clicando em File|New>Form salvando-a como "noticias.php". Em seguida copie a imagem, os *Label's* de menu e o *label* de data para a nova página. Repita a programação do evento *OnShow* do *label* de data nessa página. Podíamos ter criado um arquivo a parte e usado *frames* para evitar a programação repetida, mas esse assunto está fora do escopo deste artigo.

Desenhe uma tela como mostrado na **Figura 7**. Aumente a largura do *label Notícias do Dia* para que fique quase do tamanho todo da tela. Note que no centro do formulário inclui um *label* aumentando sua largura e altura de forma que ocupe todo o espaço da tela. Em sua propriedade *Caption* colei um texto da internet bastante grande. Isso irá simular uma notícia extensa. Perceba também que inseri novamente um componente *StyleSheet*. Ele precisa ser configurado

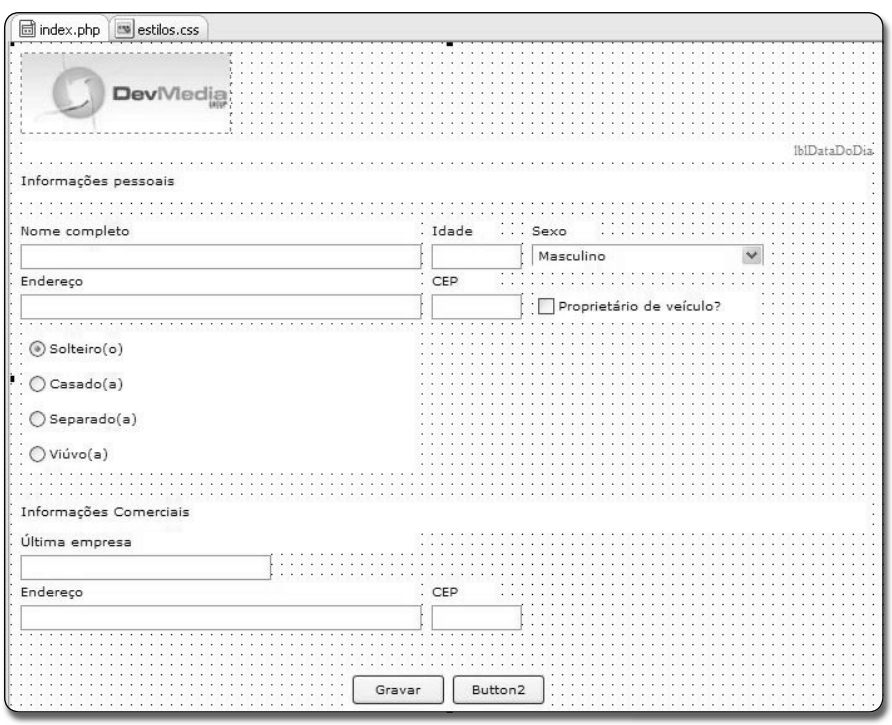


Figura 1. Formulário de cadastro para uso com CSS

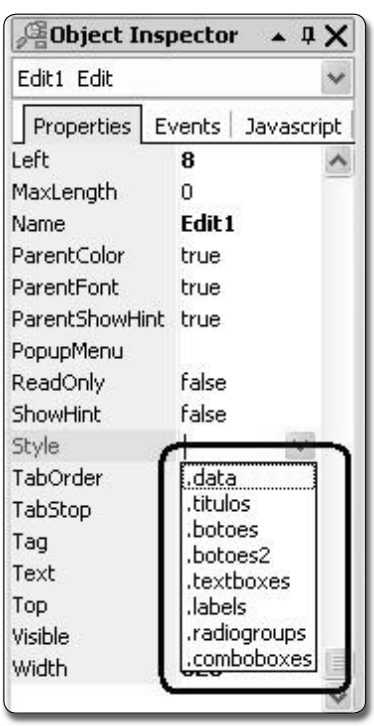


Figura 2. Seleção das regras do CSS

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Adriano Santos que mostra como trabalhar com frames no Delphi for PHP.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=6374

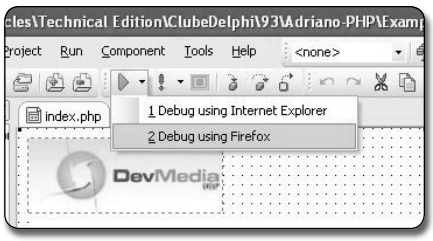


Figura 4. Como executar o teste em mais de um browser

Listagem 3. Código CSS completo do exemplo

```
.data{
        font-family: Trebuchet MS;
        text-align: right;
        color: #FF0000;
        font-size: 11px;
        background-color: #fff;
}
.titulos{
        font-family: Trebuchet MS;
        text-align: center;
        color: #fff;
        font-size: 20px;
        background-color: #AE5700;
}

.botoes{
        color: #fff;
        font-size: 11px;
        font-family: Trebuchet MS;
        background-color: #4DA4D2;
        border-width: 1px;
        border-style: dashed;
        border-color: #000;
}

.textboxes{
        color: #4DA4D2;
        font-size: 11px;
        font-family: Trebuchet MS;
        background-color: #fff;
        border-width: 1px;
        border-style: dashed;
        border-color: #4DA4D2;
}

.labels{
        color: #4DA4D2;
        font-style: bold;
        font-size: 11px;
        font-family: Trebuchet MS;
        background-color: #fff;
}

.radiogroups{
        color: #4DA4D2;
        background-color: #fff;
        border-width: 1px;
        border-style: solid;
        border-color: #4DA4D2;
        font-size: 11px;
        font-family: Trebuchet MS;
}

.comboboxes{
        color: #4DA4D2;
        background-color: #fff;
        border-width: 1px;
        border-style: dashed;
        border-color: #4DA4D2;
}
```

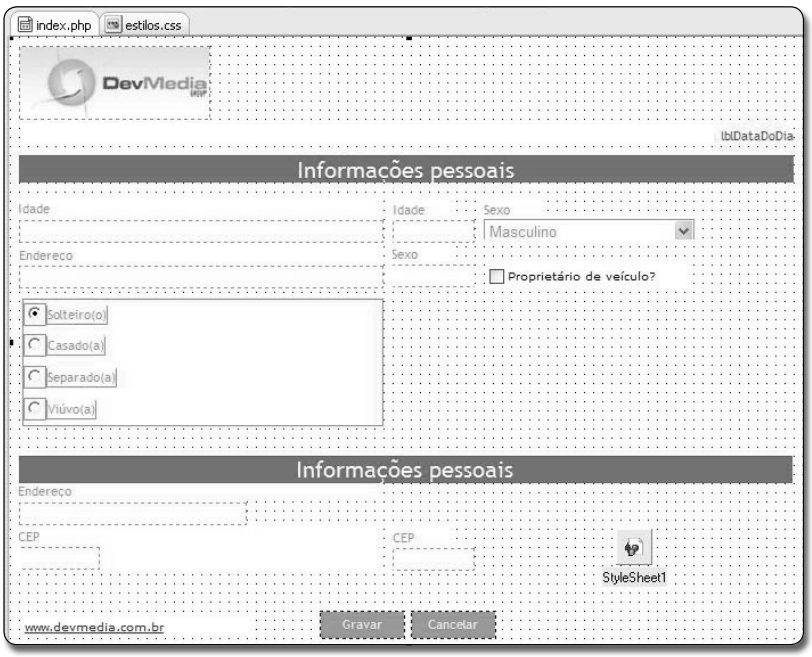


Figura 3. Layout com CSS visualizado no Delphi for PHP

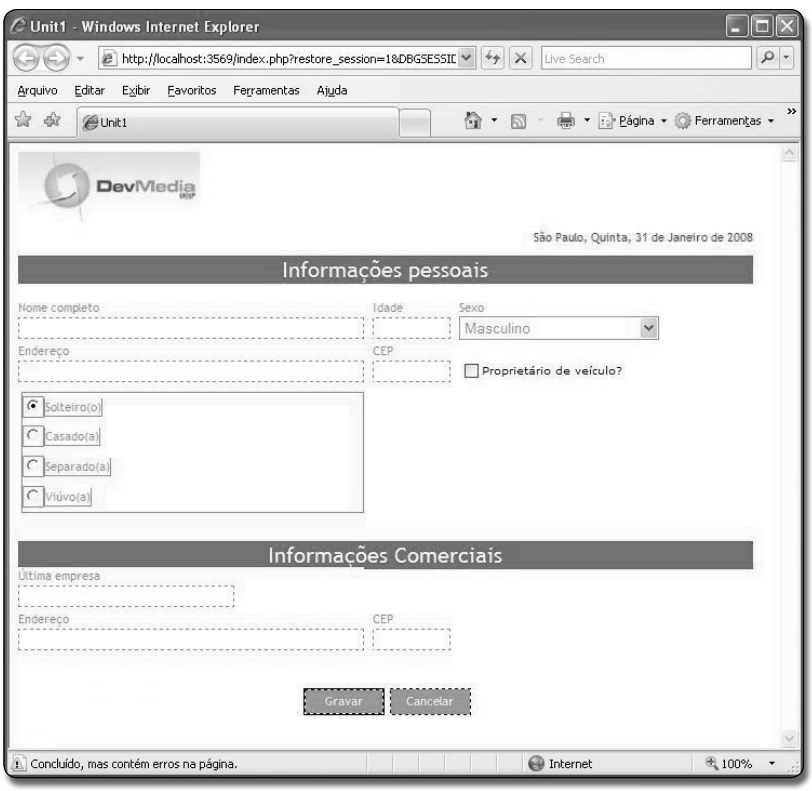


Figura 5. Teste da página no Internet Explorer

Figura 6. Link sob o mouse

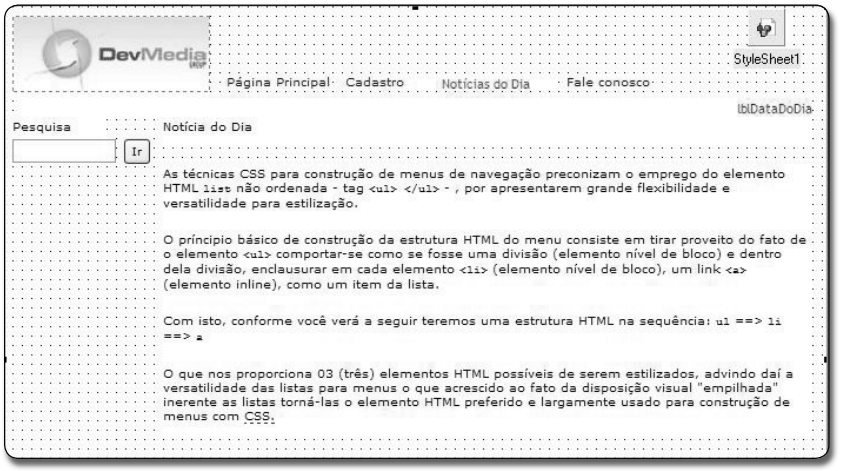


Figura 7. Página de notícias sem CSS

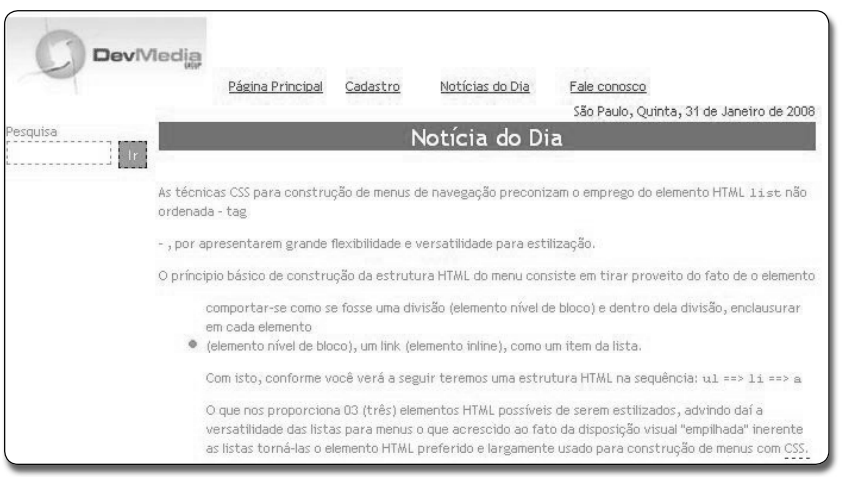


Figura 8. Página de notícias com CSS aplicado

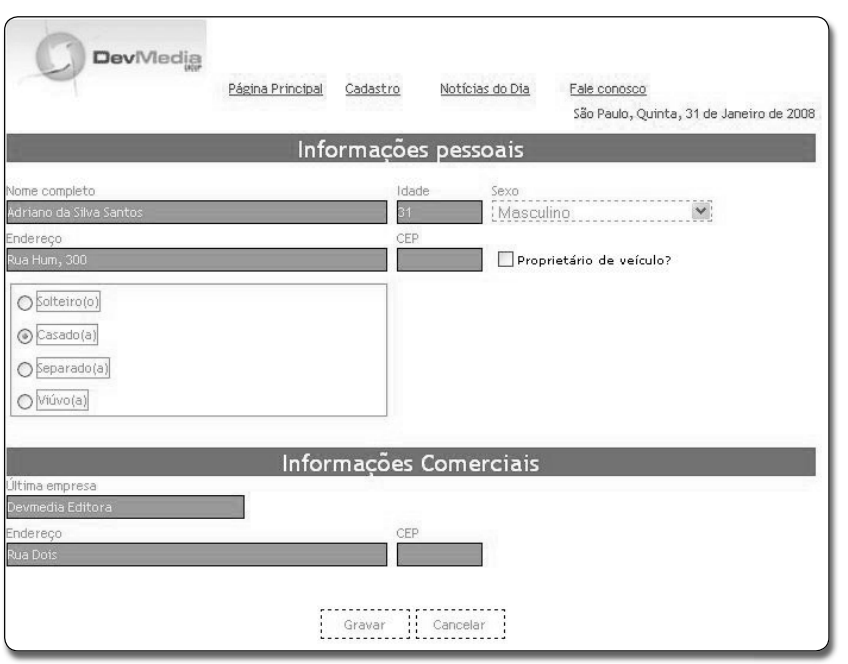


Figura 9. Layout final depois de alterado

Listagem 4. Inclusão de regras para links

```
a:link {
        color: #4DA4D2;
        background-color: #fff;
        font-size: 11px;
        font-family: Trebuchet MS;
}
a:visited {
        color: #FF0000;
        background-color: #fff;
        font-size: 11px;
        font-family: Trebuchet MS;
}
a:hover {
        text-decoration: underline;
        color: #fff;
        background-color: #4DA4D2;
        border-width: 1px;
        border-style: dashed;
        border-color: #000;
}
```

como fizemos nos passos anteriores.

Agora basta fazer a ligação de cada controle com suas respectivas regras CSS. Não esqueça de digitar na propriedade *Link* dos *label "Página principal"* e *"Notícias do dia"* o endereço da página que será aberta. Volte para a página principal e execute-a. Experimente clicar em *"Notícias do dia"* e veja o resultado (**Figura 8**).

Modificando o layout do site

Depois de feitas todas as alterações necessárias, vinculado cada controle a suas respectivas regras podemos ficar tranqüilos quanto aos padrões do nosso sistema. Agora toda vez que precisarmos fazer qualquer alteração, basta acessarmos novamente o arquivo *estilos.css*, alterá-lo e enviá-lo novamente ao servidor FTP. Experimente inverter as cores do fundo e fonte da regras *.botoes* e executar novamente o site. Perceberá nitidamente que todos os botões vinculados a essa regra forma automaticamente modificados, sem a necessidade de alterar página a página (**Figura 9**).

Conclusão

O Delphi for PHP ainda está na versão 1.0, mas dá sinais de que será uma grande ferramenta. Nesse artigo aprendemos a usar um dos recursos mais utilizados no dia-a-dia. Separamos o código, onde possíveis regras de negócios poderiam estar implementadas, da parte visual, ou seja, o layout. Ainda é possível trabalharmos com templates e frames agilizando ainda mais o desenvolvimento. Lembre-se que o limite é a criatividade. Boa sorte e até a próxima. ●

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet

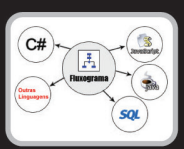
Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

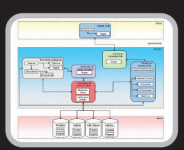
Simplificamos a tecnologia.

Java[4] e .Net[1]

Você cuida da lógica e o Maker faz o resto. É ele quem interage com a camada de complexidade, garantindo que seu sistema funcione com perfeição. Com desenvolvimento até 60x mais rápido, vai sobrar tempo para você aprimorar ainda mais suas aplicações.

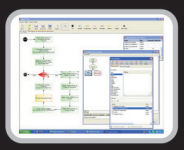


O Maker permite abstrair o desenvolvimento de aplicações deixando para o desenvolvedor apenas a preocupação com a lógica do negócio, pois é o Maker que interage com as linguagens como **JAVA, .Net, JavaScript, C++, TransactSQL, PL/SQL** e **C#**.

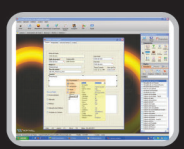


Todos os sistemas criados com o Maker têm uma arquitetura MVC.

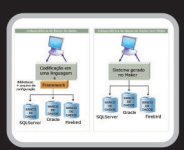
Sistemas criados com o Maker são **stateless** garantindo assim a performance e a escalabilidade das aplicações.



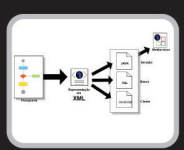
A criação de rotinas de software com o Maker é feita de forma totalmente visual, através do uso de fluxogramas, tornando-a mais intuitiva, permitindo alta produtividade e facilitando a manutenção.



O Maker dispõe de diversos assistentes e mecanismos para agilizar o processo de desenvolvimento, tais como primitivas do RAD e outros.



O Maker possibilita a total independência de bancos de dados, permitindo um desenvolvimento rápido, barato e com alta performance. O Maker é capaz de interagir com os bancos: **Oracle, SQL Server, MySQL, Firebird, Postgre e DB2.**



Os sistemas criados com o Maker estão numa camada acima das linguagens de programação, garantindo a evolução das aplicações independente da tecnologia aplicada.

Todas as aplicações desenvolvidas com o Maker são para ambientes web, sem a necessidade de instalação de applets nas estações de trabalho, sem abrir mão dos recursos disponíveis para o ambiente desktop.